CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL RHÔNE-ALPES

CENTRE D'ENSEIGNEMENT DE GRENOBLE

Mémoire présenté par

Eric Callegari

en vue d'obtenir

Le Diplôme d'Ingénieur C.N.A.M.

En Informatique

La continuité de services en informatique ambiante : conception et adaptation d'applications sur WComp.

Soutenu le 14 janvier 2009, à Grenoble, devant le jury :

Présidente :	Mme Véronique Donzeau-Gouge (CNAM Paris)
Examinateurs :	M. Eric Gressier-Soudan (CNAM Paris)
	M. Jean-Pierre Giraudin (CNAM, UPMF Grenoble)
	M. André Plisson (CNAM Grenoble)
	M. Matthias Voisin-Fradin (CNAM Grenoble)
Tuteurs :	M. Jean-Yves Tigli (Université Nice Sophia Antipolis)
	M. Stéphane Lavirotte (Université Nice Sophia Antipolis)

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL RHÔNE-ALPES

CENTRE D'ENSEIGNEMENT DE GRENOBLE

Mémoire présenté par

Eric Callegari

en vue d'obtenir

Le Diplôme d'Ingénieur C.N.A.M.

En Informatique

La continuité de services en informatique ambiante : conception et adaptation d'applications sur WComp.

Les travaux relatifs au présent mémoire ont été effectués sous la direction de Jean-Yves Tigli, au sein de l'équipe de recherche Rainbow, dirigée par Michel Riveill. Rainbow est intégrée au laboratoire I3S, UMR (Unité Mixte de Recherche) du CNRS.

REMERCIEMENTS

Pour la réalisation de ce mémoire et la possibilité de le faire, je tiens à remercier :

- M. Jean-Yves TIGLI, maître de conférences à l'école polytechnique de l'Université de Nice-Sophia Antipolis, mon maître de stage, qui a toujours été disponible tout au long de cette année de stage, pour m'aider dans mes travaux mais également dans ma recherche d'emploi.
- M. Michel RIVEILL, directeur du département "Systèmes informatiques" à l'école polytechnique de l'Université de Nice-Sophia Antipolis, responsable de l'équipe "Rainbow", pour avoir accepté de m'accueillir au sein de son équipe.
- MM. Stéphane LAVIROTTE et Gaétan REY, maîtres de conférence à l'Université de Nice-Sophia Antipolis, pour leur aide tout au long du stage.
- M. Jean-Pierre GIRAUDIN, responsable pédagogique et scientifique du cycle ingénieur CNAM-Informatique à Grenoble, pour son aide dans ma recherche de stage.
- Le CNAM de manière générale et tous les enseignants sur mon parcours en particulier, qui m'ont permis de poursuivre mes études.

Par-dessus tout, je tiens à remercier mon épouse Nathalie pour son soutien constant et pour l'ambition qu'elle m'a inspirée.

SOMMAIRE

TABLE DES ILLUSTRATIONS

INTRODUCTION

Partout où des services sont mis à disposition d'utilisateurs, la question de la continuité de services se pose. En particulier, elle est un thème essentiel du domaine des réseaux informatiques. En effet, quel que soit le service proposé, son fournisseur est désireux d'en assurer la disponibilité constante à travers tout le réseau qui l'héberge. Dans une informatique qui voit la multiplication de nouveaux services innovants pour l'utilisateur, la fiabilité et la disponibilité ne doivent pas être compromises. Dans une première approche, la continuité de services se traduit par l'adaptation de l'application aux changements survenant sur les équipements qui constituent le réseau.

Cependant, l'avènement de l'informatique dite mobile et/ou ambiante ne permet plus de concevoir des applications logicielles dédiées à des plateformes prédéfinies et composées de dispositifs connus à priori, comme c'est le cas dans un réseau classique. En effet, les nouvelles applications doivent non seulement gérer une grande hétérogénéité des dispositifs composant le réseau, mais aussi s'adapter à l'utilisateur ou à l'environnement. Par conséquent, il existe une autre manière de poser la problématique de la continuité de services, les logiciels étant tenus de s'adapter à une multitude de contextes d'utilisation. Dans ce cadre quelle est la signification de la notion de contexte ? Quels sont alors les concepts et mécanismes qui permettent à une application de s'adapter à son contexte ? En quoi la plateforme WComp répond-elle aux besoins de l'informatique ambiante ?

Le présent rapport peut se décrire en deux phases. La première définit ce qu'est l'informatique ambiante, décrit les notions de continuité de services et de contexte. Cette première phase se termine par la présentation des divers paradigmes mis en œuvre en informatique ambiante. La deuxième phase commence par l'examen des différentes plateformes existantes permettant l'adaptation des applications à leur contexte ainsi que du modèle de la plateforme WComp. Les parties 5 et 6 décrivent l'aspect expérimental du sujet, présentant la plateforme, les mécanismes et moyens mis en œuvre pour l'adaptation des applications à leur contexte, ainsi que les difficultés rencontrées et les moyens de les contourner.

1 Qu'est ce que l'informatique ambiante ?

1.1 Définition

Dans notre environnement, l'informatique est partout : PC, téléphones portables, PDA, équipements de voitures, distributeurs automatiques en tout genre, … On peut donc affirmer que l'informatique aujourd'hui est "ambiante" au sens propre du terme.

Mark WEISER[1], qui est considéré comme le père du concept d'informatique ambiante, a déclaré : "*Silicon-based information technology, is far from having become part of the environment* "[1], que l'on peut traduire par "l'informatique est encore loin de faire partie de l'environnement". Cette déclaration démontre bien le véritable objectif de l'informatique ambiante, qui est de faire oublier l'informatique dans notre quotidien ou en d'autres termes, de diluer l'informatique dans notre environnement, notamment dans des objets qui n'ont à priori aucune fonction informatique. A cet égard, le concept d'informatique ambiante va plus loin que le simple envahissement de nos espaces de vie par les ordinateurs de toutes sortes.

On peut décrire l'informatique ambiante de différentes manières. Beaucoup de termes existent : on parle d'informatique omniprésente ("*ubiquitous computing*" en anglais), ou encore d'informatique "*pervasive*". Toutes ces notions gravitent autour du même concept qui est de vouloir sortir les applications informatiques du cadre de l'ordinateur classique. Par exemple, pourquoi ne pas visionner sur sa télévision les courriels qu'on a reçus ou encore, dans un tout autre genre, pourquoi ne pas permettre au vase de communiquer à propos de l'ensoleillement ambiant ou de l'état de l'eau qu'il contient, etc.

Dernièrement, une nouvelle expression a fait son apparition : l' "*everyware*". Ce mot nouveau, inventé par Anthony Townsend[2], regroupe tous les termes cités plus haut et exprime remarquablement bien l'omniprésence du logiciel dans notre vie de tous les jours. Il est néanmoins très générique. En effet, il englobe :

- L'informatique classique et l'informatique mobile ;
- Les réseaux comme internet mais aussi les autres technologies de connexion informatique ;
- La réalité augmentée ;
- La domotique et les véhicules intelligents ;
- Les technologies de l'internet des objets comme les RFID par exemple [2].

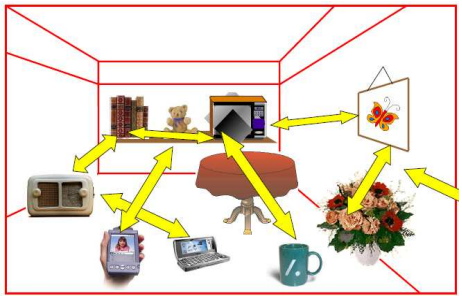


Figure 1 : **informatique ambiante ou omniprésente**

[1] Mark Weiser : directeur de la recherche à Xerox, 1952-1999

[2] Anthony Townsend : directeur de recherche à "l'Institut pour le Futur", Palo Alto (Californie).

L'informatique ambiante introduit un nouveau mode d'utilisation. En effet, tandis que l'informatique classique est concentrée en un seul endroit et relie un service à un appareil, elle permet à un utilisateur d'interagir simultanément avec un grand nombre d'appareils et de services, ces derniers étant enfouis dans les divers objets de notre quotidien. De plus, l'interaction ne se fait plus seulement grâce aux menus, pointeurs, souris et autres accessoires de l'ordinateur, mais aussi par la voix, le toucher,... Enfin, l'usage d'un ordinateur classique résulte toujours d'une décision consciente de l'utilisateur. L'informatique ambiante s'utilise sans y penser, sans le savoir et même parfois sans le vouloir [2]. L'informatique ne s'utilise plus, elle se vit.

1.2 Quelques exemples

1.2.1 Nabaztag

Nabaztag (signifie "lièvre" en arménien) est un objet communicant de 23 cm de haut qui représente un lapin. Il est capable de se connecter à internet par ondes WI-Fi 802.11b. Il communique avec son utilisateur en émettant des messages vocaux, lumineux ou en remuant les oreilles. Il diffuse des informations du type météo, bourse, circulation routière, arrivée de courriels etc.

Figure 2 **: Nabaztag, un objet communicant**

Le Nabaztag a cette particularité qu'il ne se fond pas dans notre environnement, à moins de posséder un lapin en plastique dans la cuisine depuis toujours ; il vient s'ajouter à l'environnement de son propriétaire pour indiquer l'arrivée de messages électroniques, donner la météo, allumer une radio, mais aussi donner son humeur et bouger les oreilles ! D'autres fonctionnalités existent, comme par exemple la gestion des flux RSS.

L'utilité d'un Nabaztag est discutable mais c'est une belle tentative pour faire sortir l'informatique du cadre du PC.

1.2.2 Le rétroviseur *Bluetooth*

Plusieurs modèles de rétroviseur bluetooth existent. Ce type de dispositif permet de téléphoner en main libre conformément aux réglementations de sécurité routière.

La conception sans fil facilite l'installation car elle ne nécessite aucune modification d'intérieur dans le véhicule. Il suffit de fixer le rétroviseur à l'aide de clips et de synchroniser le mode *bluetooth* du téléphone portable.

Lors d'un appel entrant le numéro s'affiche au bas du rétroviseur ; l'utilisateur a alors deux possibilités :

- Utiliser une oreillette *bluetooth*,

- Ecouter son interlocuteur sur le haut-parleur embarqué.

Figure 3 **: Le rétroviseur Bluetooth**

L'utilité d'un tel dispositif est indéniable.

1.2.3 Le cadre photo numérique

De nouveaux modèles WiFI existent. Ils peuvent se connecter à internet pour télécharger de nouvelles images.

Figure 4 **: Le cadre numérique**

Cet objet diffère des deux précédemment décrits. En effet, ces derniers visent tous à fournir à l'utilisateur un service déjà existant ; le nabaztag permet entre autres la réception de courriels même en l'absence de PC et le rétroviseur *bluetooth* permet de recevoir des appels sans toucher à son téléphone portable. Le cadre numérique, quant à lui, introduit un nouveau service, ou du moins améliore un service existant en permettant un changement automatique d'image dans un cadre.

On peut néanmoins s'interroger sur l'utilité de ce nouveau service, particulièrement à la vue du prix du cadre.

1.2.4 Microsoft *Surface*

Microsoft a dévoilé (mai 2007) une version finale de son prototype d'écran tactile de grande dimension, baptisé *Surface*. L'utilisateur peut interagir avec la machine en touchant l'écran ou en glissant ses doigts ou des objets comme des pinceaux sur sa surface, ou encore en y plaçant des objets ordinaires sur lesquels figurent des codes barres spéciaux [3].

Figure 5 **: Surface, nouvel écran tactile de Microsoft**

Voilà un autre type de dispositif en informatique ambiante. On peut le classer dans les interfaces tangibles. Ce type d'interface vise à rendre l'utilisation d'un service plus naturelle ou plus intuitive. On peut classer dans cette catégorie la désormais célèbre console Wii de Nintendo.

1.3 Conclusion

Le concept d'informatique ambiante est suffisamment large pour voir émerger des dispositifs de toutes sortes, rivalisant d'originalité et d'ingéniosité. Et si certains d'entre eux semblent inutiles ou "gadget", que dire des premiers jeux sur nos ordinateurs ? Il n'est pas impertinent de penser que l'informatique ambiante suivra la même trajectoire ascendante que l'industrie du jeu video.

Certains de ces nouveaux dispositifs tentent de se fondre dans notre environnement habituel, tandis que d'autres sont totalement nouveaux. Certains visent à rendre l'informatique plus intuitive et accessible au plus grand nombre. Enfin, certains d'entre eux fournissent de nouveaux services, alors que d'autres assurent la continuité de services existants.

2 La continuité de service en informatique ambiante

2.1 Première approche

A l'origine utilisée dans le domaine des services telecom, la notion de continuité de services est définie comme étant un mécanisme qui permet à un utilisateur de garder de manière transparente et sans intervention manuelle un service actif sur la zone de couverture de ce service. Plus précisément, on parle de disponibilité de services.

La disponibilité de services se décline en deux volets :

- La fiabilité et la disponibilité des composants hardware ;
- La continuité du service.

La notion de continuité de services concerne la gestion des communications entre le service informatique (ou application) et l'infrastructure logicielle, qui est définie comme étant l'ensemble des équipements pouvant interagir avec l'application. Dans cette première approche de la problématique, l'infrastructure logicielle n'est composée que des équipements du réseau (fig.6).

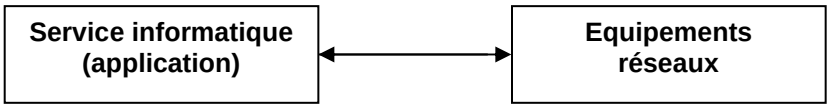


Figure 6 : **modèle d'un système informatique en réseau**

Que devient la problématique si l'on considère maintenant des services en informatique ambiante ?

2.2 Aller plus loin

Le cadre de l'informatique ambiante est celui de l'interaction de trois familles d'entités : systèmes informatiques, environnement et utilisateur(s). En informatique ambiante, le modèle du système n'est plus seulement constitué des deux entités définies sur la figure 6. Le modèle d'un tel système est le suivant :

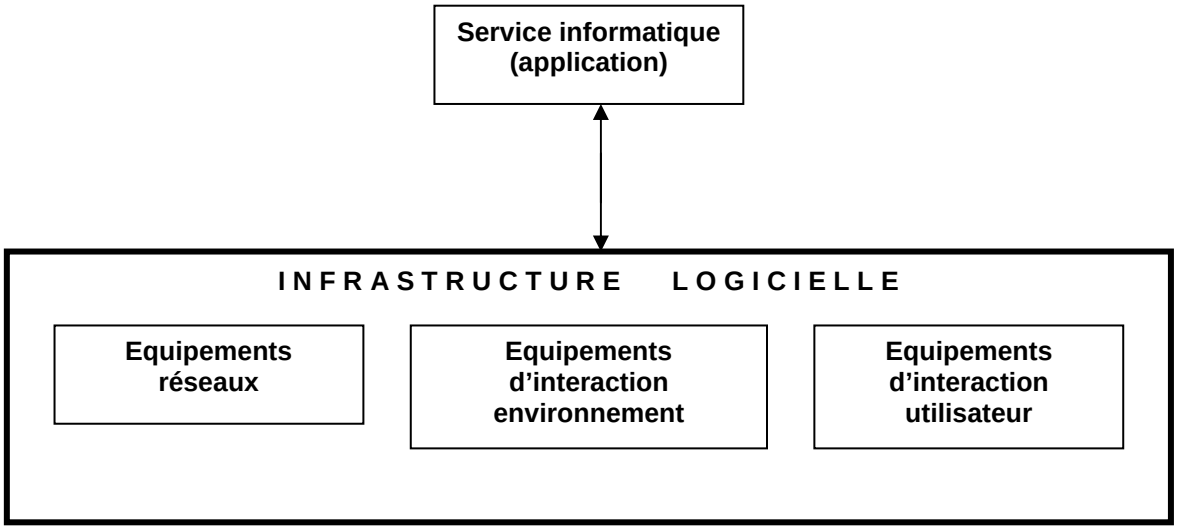


Figure 7 **: modèle d'un système en informatique ambiante**

On considère ici une infrastructure logicielle enrichie. En effet, on ne s'intéresse plus seulement à l'interaction de l'application avec des équipements connus et définis au départ mais aussi avec des équipements qui interagissent eux-mêmes avec le monde physique ou avec l'utilisateur et qui se caractérisent par leur grande hétérogénéité.

Ces équipements d'interaction avec l'utilisateur ou l'environnement vont permettre au service d'être utilisable à tout moment, sous une forme ou une autre. Par exemple, un automobiliste pourra téléphoner dans son véhicule grâce à des équipements de reconnaissance et de synthèse vocale ainsi que des capteurs détectant sa situation (dans le véhicule ou non).

Le fait de capter le réseau pour pouvoir téléphoner est essentiel mais c'est une sous problématique. Le but est de rendre le service "téléphonie" utilisable quelle que soit la situation de l'utilisateur et quel que soit son environnement. On parle du contexte de l'utilisateur.

2.3 La notion de contexte

La notion de contexte est utilisée dans beaucoup de domaines, comme par exemple en intelligence artificielle, en linguistique, en psychologie et bien sur, en informatique ambiante [4].

Dans les premiers travaux sur le sujet, le contexte était souvent associé à la localisation dans l'espace. En ce qui concerne l'informatique ambiante, il n'existe toujours pas de définition consensuelle de cette notion, mais de manière générale, le contexte désigne toute information qui peut être utilisée pour caractériser une entité. Il s'agit d'un ensemble structuré d'informations qui peuvent concerner :

- L'environnement : les personnes ou objets à proximité, le bruit, le climat, etc.
- L'utilisateur : sa localisation, ses centres d'intérêts, ses émotions, etc.
- La machine : serveurs à proximité, occupation des ressources, dispositif d'affichage, etc.
- Le temps : historique des actions, des localisations, date et heure du système, etc. [5]

En fait, définir le contexte ne constitue pas une fin en soi. Ce n'est qu'un moyen d'étudier autre chose. En conséquence, la définition proposée, c'est à dire la nature des informations qu'il contient ainsi que leur interprétation, seront au service de la finalité de l'étude que l'on veut mener. En informatique ambiante, il existe deux finalités qui poussent à modéliser le contexte :

- L'étude de l'adaptation de l'application à l'utilisateur nécessite que l'on modélise le **contexte d'interaction**. On utilise ce contexte dans le cadre d'une vision IHM du problème. Il s'agit de permettre à l'application de s'adapter à la situation de l'utilisateur. L'objectif est de maintenir l'utilisabilité de l'application. On parle également de "contexte utilisateur".

- L'étude de l'adaptation de l'application à la dynamique de ses composants nécessite que l'on définisse le **contexte d'exécution**. En ce qui concerne ce contexte, on s'intéresse à l'aspect système du problème, l'application devant s'adapter aux changements survenant au sein de l'infrastructure logicielle. Il s'agit ici de maintenir les fonctionnalités de l'application.

Le présent rapport traite de l'adaptation des applications au contexte d'exécution. Il s'agit de garantir les fonctionnalités et la disponibilité des applications, quelle que soit l'évolution dynamique de l'infrastructure informatique. L'aspect IHM du problème n'est pas traité ici.

2.4 Contexte d'exécution

Lorsque l'on s'intéresse au contexte d'exécution, l'objectif final est de permettre à l'application de gérer les situations externes qui influencent sa qualité de service vue par l'utilisateur. Par conséquent, l'application doit pouvoir s'adapter aux disparitions et apparitions de dispositifs sur le réseau par exemple, ou encore à toutes sortes de défaillances techniques, malveillances ou charge inhabituelle.

Le contexte n'intervient pas directement dans l'application, mais il permet de définir **l'environnement d'exécution** de l'application qui est un sous-ensemble de l'infrastructure logicielle. Il s'agit en fait des ressources disponibles du système à un instant donné (fig.8).

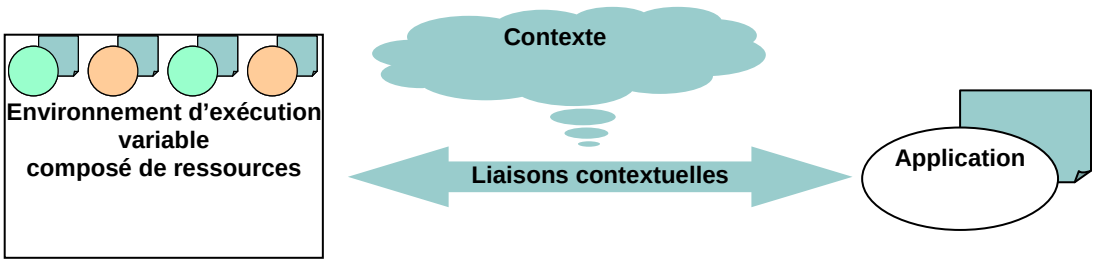


Figure 8 : **contexte et environnement d'exécution**

Une application peut avoir accès à un grand nombre de dispositifs de l'infrastructure. Ces accès doivent être supervisés au travers de la mise en place de liaisons temporaires. Ces dernières sont non seulement soumises aux capacités de communication du système, mais aussi à la validation de conditions contextuelles qui autoriseront ou non les dispositifs à

collaborer. Ces liaisons sont appelées des liaisons contextuelles [4]. L'application adaptera le service en fonction des ressources disponibles, c'est à dire de l'environnement d'exécution.

Par exemple, pour un service donné, l'application doit s'adapter à la présence ou non d'un dispositif d'affichage dans l'environnement de l'utilisateur, selon que ce dernier soit chez lui, dans la rue ou dans sa voiture. Ou encore, comment préserver pour un automobiliste la possibilité de communiquer par l'intermédiaire de son téléphone portable sachant que l'IHM de ce dernier ne fait pas partie de l'environnement d'exécution de l'application ? Une autre condition contextuelle, non liée à la présence ou non de dispositifs sur le réseau celle ci, pourrait être l'impossibilité d'allumer un ordinateur à partir d'une certaine heure, dans le but, par exemple, d'imposer un contrôle parental.

2.5 Déclenchement de l'adaptation

Les divers concepts et paradigmes utilisés pour l'adaptation des applications à leur contexte seront examinés plus loin. La question du déclenchement de l'adaptation est importante. En effet, qui va initier l'adaptation de l'application à son contexte ? Deux options sont possibles :

- Le système s'adapte automatiquement sans intervention de l'utilisateur. On parle alors d' "*autonomic computing*" ou de "*ressource-aware computing*".
- L'utilisateur intervient pour faire évoluer l'application : on introduit alors la notion de "*end-user programming*".

2.5.1 Autonomic computing

Il s'agit au départ d'une initiative d'IBM en 2001. Son objectif est de créer des systèmes informatiques capables de s'autogérer, de surmonter les difficultés de gestion liées à des systèmes toujours plus complexes, et de s'affranchir des problèmes que cette complexité pose à l'égard de la croissance future de ces systèmes.

L'*Autonomic computing* est un modèle informatique d'autogestion, qui est nommé et construit sur le modèle du système nerveux autonome du corps humain. Un tel système doit être capable de contrôler le fonctionnement d'applications sans aucune action de l'utilisateur, de la même manière que notre système nerveux gère notre corps sans décision consciente de notre part. De cette manière, un système fonctionne "tout seul" et masque sa complexité à l'utilisateur.

2.5.2 Intervention de l'utilisateur

Du point de vue de l'utilisateur, l'adaptation automatique des logiciels présentent trois risques :

- Interruption inattendue de l'activité,
- Inadéquation de l'adaptation,
- Incompréhension de l'adaptation.

Ces risques peuvent être maîtrisés en introduisant le contrôle de l'utilisateur. Ce dernier couvre trois niveaux d'exigences croissantes : suivre l'évolution du processus d'adaptation, intervenir dans ce processus, initier le processus, c'est à dire avoir la possibilité de programmer.

L'adéquation des choix du système n'est possible que si ce dernier possède un modèle avancé de l'utilisateur. En d'autres termes, le système prendra les bonnes décisions s'il "connaît" très bien l'utilisateur ce qui n'est presque jamais le cas.

Suivre l'évolution du processus d'adaptation signifie aider l'utilisateur à comprendre la nature du changement. Des techniques d'animation et de traces graphiques ont été proposées récemment. Par exemple, l'approche "*Phosphor*" part du principe qu'un utilisateur peut ne pas remarquer un changement qui vient de s'opérer sur son écran. Lui ou un collaborateur distant peut avoir fait une erreur de manipulation comme effacer accidentellement un icône ou changer des paramètres dans un panneau de configuration. Lors d'un tel changement, des objets phosphorescents se superposent aux icônes pour garder la mémoire de l'état précédent, comme illustré figure 9 [6].

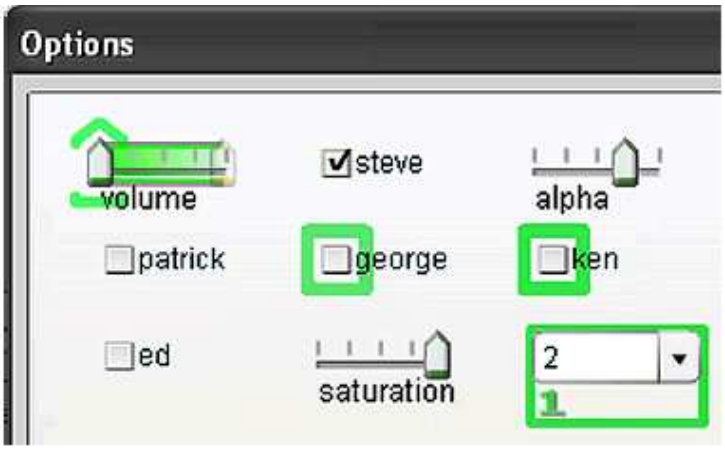


Figure 9 **: approche "phosphor"**

Mais cette approche suppose l'existence de ressources écran ce qui n'est pas toujours le cas en informatique ambiante.

La possibilité pour l'utilisateur de programmer est connue sous le nom "*end-user programming*". La programmation pour l'utilisateur final s'adresse à des non-informaticiens. Par conséquent, les langages informatiques dont on a l'habitude sont remplacés par des symboles, plus faciles à manipuler. Les travaux de recherche sur le sujet ont pour objectif la définition de notations simplifiées. Parmi les principales techniques utilisées, on trouve la programmation par l'exemple et la programmation visuelle. On parle aussi de programmation naturelle.

La **programmation par l'exemple** ou programmation par démonstration part du constat que l'utilisateur est très souvent en train de répéter les mêmes tâches. L'idée est d'apprendre à l'ordinateur de nouveaux comportements par la démonstration. Le système enregistre les séquences d'actions de l'utilisateur et génère un programme qui pourra être utilisé ultérieurement.

Les langages de **programmation visuelle** sont des langages qui permettent à l'utilisateur de programmer avec des symboles graphiques. La plupart d'entre eux sont constitués de boîtes connectées entre elles par des flèches. Les langages de programmation visuelle sont au départ des langages textuels et qui ont une couche graphique, chaque élément symbolique ayant une traduction en code.

Le concept de **programmation naturelle** décrit des langages capables d'interpréter notre langage naturel en code exécutable. Cela fait appel à des techniques d'intelligence artificielle

qui, pour l'instant butent sur le problème de l'interprétation de nos langages courants. Par conséquent, de tels langages de programmation n'existent pas encore [7].

Parmi les langages de programmation pour utilisateur final, on compte beaucoup de langages spécifiques à des applications. Dans ce cas, bien que l'utilisateur ne soit pas un informaticien, il demeure un spécialiste de l'application et par conséquent ces langages ne sont pas accessibles à un utilisateur quelconque.

Exemple : LUSTRE et SCADE.

Le langage LUSTRE est un langage de programmation synchrone qui traite en entrée des flots de données. SCADE est la version graphique de LUSTRE [8].

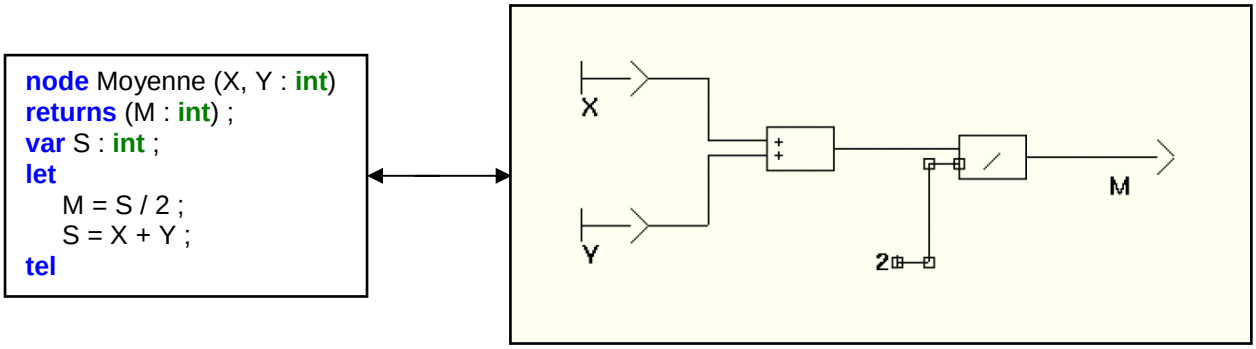


Figure 10 **: Equivalence LUSTRE-SCADE**

Après avoir examiné les différents cas de déclenchement d'une adaptation, il est légitime de se demander comment une application va s'adapter à son environnement ou à l'utilisateur. En particulier, quels sont les concepts ou paradigmes qui sous tendent l'adaptation des applications à leur contexte ?

3 Paradigmes pour l'informatique ambiante

Depuis plusieurs années, les composants et les services sont apparus comme des solutions efficaces aux problèmes de dynamicité et d'interopérabilité, et par conséquent également dans le domaine des applications en informatique ambiante.

3.1 Composants et services

3.1.1 Conception orientée composants

Les systèmes basés sur les composants sont une évolution du paradigme de programmation orientée objets. Le but de tels systèmes est de résoudre les problèmes de réutilisation du code liés à ce type de programmation. Un composant logiciel est une unité de composition avec des interfaces contractuellement spécifiées. Une application se présente alors sous la forme d'un ensemble de composants, plus précisément d'instances de types de composants, ces dernières étant liées entre elles.

Il existe différents modèles de composants :

- Les modèles de composants légers et dynamiques, par exemple, les JavaBeans, ou les .NET *components*. Par "léger", on entend que de tels composants ne contiennent pas de références vers d'autres composants au moment de la conception. Leurs atouts sont réutilisabilité, adaptabilité et dynamicité.
- Les autres modèles de composants. Par exemple, Fractal, EJB ou CCM. Ces modèles sont moins flexibles. En effet, ces composants possèdent des références vers d'autres composants (via une interface). Il est donc nécessaire de compiler le code du composant en fournissant le fichier correspondant à l'interface fournie d'un autre composant.

Certains modèles de composants permettent de définir une hiérarchie entre composants : composants et composants composites, ces derniers étant des assemblages de composants. Mais les interfaces des composants composites ne sont pas dynamiques. En effet, du code est généré lors de leur création car la structure de l'assemblage de départ est modifiée.

Les composants interagissent entre eux via leur interface, par l'intermédiaire de liaisons. Elles correspondent à des appels de méthodes locales, avec valeurs de retour (Fractal, COM). Lorsque les composants sont distribués, il s'agit d'appel de méthodes à distance, avec RMI par exemple (EJB, CCM). Dans des cas plus adaptés à l'informatique ambiante, les évènements ou passages de messages asynchrones sont utilisés : un composant émet un évènement qui déclenche une exécution sur un autre composant (JavaBeans, .NET *components*, EJB, CCM) [9]

3.1.2 Architecture orientée services (S.O.A.)

Les architectures orientée services ont apportées huit principes ou influences au domaine du génie logiciel. Tout d'abord, l'**encapsulation** des services, permettant à n'importe quel programme de s'exécuter en tant que partie d'une application ; le **couplage faible** qui

minimise les dépendances entre les services et qui augmente la dynamicité et la **réutilisabilité** ; le **contrat** de service qui oblige les services à adhérer à un standard de communication, fournissant la description de ce qu'ils offrent ou requièrent ; l'**abstraction**, également connue sous le nom d'abstraction "boîte noire", qui limite la connaissance du service au contrat ; l'**autonomie** des services, rendant les services indépendants les uns des autres, et auto-suffisants pour le service qu'ils procurent. La **découverte** des services, permettent aux services d'être découverts dynamiquement en cours d'exécution, avec des critères ; et enfin la **composition**, coordonnant les services et les assemblant pour former des services composites.

Une application à base de services est conçue comme une orchestration de services, qui définit les flots d'exécution de l'application [9]. Techniquement, on ne crée pas d'instance de service, son existence est implicite, dès lors qu'il y a interaction avec le service, et, en théorie, son état interne reste le même pour tous les clients. L'appel d'un service ne dépend donc pas des autres services interagissant dans l'application. Pour les composants, en revanche, la création d'instance est explicite et les états internes sont libres d'évoluer.

L'approche orientée service possède des points communs avec l'approche de programmation orientée composant. Dans les deux cas, il y a séparation des fonctionnalités en différentes entités et utilisation des fonctionnalités par le biais d'interfaces. Néanmoins, les fonctionnalités offertes par les services sont en général de plus gros grain que celles des composants et les services sont plus adaptés à la programmation distribuée.

Pour une meilleure réutilisabilité, il serait souhaitable de voir la programmation orientée composants adopter le concept de logiciel "boîte noire". En effet, dans le cadre d'un modèle hiérarchique de composants, les composants composites sont au contraire des "boîtes blanches", c'est-à-dire qu'ils offrent la possibilité de modifier leur structure interne, et donc d'accéder à leur implémentation, dans le but d'adapter leur comportement.

L'utilisation d'une architecture orientée services dans la cadre d'un système comprenant des dispositifs d'entrée-sortie vers l'utilisateur et l'environnement physique est tout aussi pertinente que la programmation orientée composants. Elle nécessite néanmoins l'extension du modèle de service de l'architecture.

3.2 Evolution des S.O.A.

Les SOA sont adaptées à la conception d'applications incluant des dispositifs (physiques ou virtuels). En effet, tout comme les services, les dispositifs ne dépendent pas des autres et ils fournissent un ensemble de fonctionnalités. En informatique ambiante, l'utilisateur évolue dans un environnement constitué de nombreux dispositifs mobiles. Or ces dispositifs ne sont pas toujours disponibles et les moyens de communiquer avec eux évoluent. Pour que ces changements puissent être pris en compte, il est nécessaire de faire évoluer le modèle des architectures orientées services. Ci-après sont examinées les principales évolutions des S.O.A., qui répondent aux besoins de l'informatique ambiante.

3.2.1 Interopérabilité

Les **Web Services** sont une première évolution du m.odèle SOA. Ils sont apparus dans un but d'interopérabilité, utilisant les technologies web comme TCP/IP, HTTP, XML pour décrire un service et communiquer avec. Ces technologies sont présentes sur la plupart des systèmes d'exploitation et dans la plupart des langages, ce qui rend ces services facilement interopérables [9].

Les *Web Services* constituent une approche pour mettre en œuvre le paradigme de service dans un contexte technologique facilitant l'interopérabilité des services indépendamment des plateformes sous-jacentes. Utiliser cette approche pour accéder à des dispositifs permet de gérer l'apparition ou de la disparition de dispositifs comme l'arrivée ou le départ d'un service. Deux extensions majeures sont alors nécessaires :

- un mode de communication par événements garantissant un minimum de réactivité aux variations des entrées/sorties des dispositifs,
- la recherche et découverte dynamique de nouveaux services pour dispositifs dans l'environnement proche de l'utilisateur.

3.2.2 Communications évènementielles

Les interactions et communications entre services ont beaucoup évolué. Traditionnellement, ce sont des appels de méthodes à distance (comme RPC). Pour les Web Services, les invocations se font avec le protocole SOAP, en XML, sur HTTP le plus souvent. Lorsqu'un service invoque une méthode sur un service distant, il suspend son flot de contrôle jusqu'à la réception de la réponse avec des valeurs de retour éventuelles. Cependant, il existe des invocations asynchrones de méthodes, permettant de continuer l'exécution du service appelant, avec une notification d'un retour de réponse.

Les communications par appel de méthodes conviennent aux applications de traitement de données ou aux services d'informations. En informatique ambiante, les services doivent pouvoir notifier leurs utilisateurs (humains ou d'autres services) d'un changement d'état des données ou de l'environnement. Ce besoin de réactivité dans les applications a impliqués l'utilisation des évènements mis en œuvre grâce à des mécanismes de publication/souscription ou de notification d'évènements [9].

3.2.3 Découverte dynamique répartie

Les modèles de services les plus anciens reposent sur une architecture centralisée, comme par exemple le broker de CORBA ou le répertoire UDDI de Web Services. Lorsqu'ils deviennent disponibles, les services s'enregistrent auprès d'un répertoire pour que les clients puissent les découvrir et les utiliser. Les consommateurs de services pourront alors effectuer une recherche sur le répertoire suivant d'éventuels critères et trouver les services aptes à être utilisés pas l'application.

Les mécanismes utilisés pour la découverte de services dans un environnement pervasif proposent le plus souvent de ne pas utiliser de répertoire de services. En effet, l'ensemble des services étant fixe, il est difficile d'en ajouter dynamiquement ou de les faire prendre en compte par les applications en cours d'exécution. S'appuyer sur une architecture fixe lorsque l'application ou l'utilisateur sont mobiles, et que les entités apparaissent ou disparaissent rapidement, n'est pas une solution optimale, puisque le répertoire lui-même peut être hors de portée. Des solutions de **recherche distribuée** ou répartie sont alors mises en place, en se basant sur des diffusions de messages à l'attention des services [9].

3.3 Services pour dispositifs

Les services pour dispositifs incluent des mécanismes comme les communications évènementielles et la découverte dynamique répartie des services [10]. Ils reprennent d'autre part les notions principales nécessaires à la communication avec des dispositifs dans des applications d'informatique ambiante, sauf l'interopérabilité. Par exemple, avec Jini, le langage Java est nécessaire pour concevoir les services. Pour bénéficier d'une part de l'interopérabilité et de la standardisation apportées par les Web Services et d'autre part de la réactivité et de la décentralisation introduites par les services pour dispositifs, les Web Services pour dispositifs ont été créés.

La notion de ***Web Services* pour dispositifs** regroupe les approches basées sur des standards adaptés de *Web Services* pour l'accès à ces dispositifs, tels qu'UPnP et DPWS.

Les premiers travaux sur les Web Services pour dispositifs ont été initiés par le consortium UPnP (*Universal Plug and Play*) en 1999. Depuis le W3C a publié un certain nombre de recommandations.

UPnP [11] est la première implémentation des *Web Services* pour Dispositifs (*Web Services for Devices*), bien qu'elle ne se définisse pas ainsi. Ce standard a été créé en 1999 par Microsoft et Intel, à partir de technologies en partie existantes et fiables comme IP, UDP et TCP, HTTP, XML, SOAP, GENA (*General Event Notification Architecture*), HTTPU et HTTPMU (http sur UDP et MulticastUDP). Elle a ensuite été développée par le forum UPnP que plusieurs centaines d'entreprises ont rejoint [12].

Le premier but d'UPnP était l'extension des notions de Plug'n'Play à tout dispositif du réseau local. Les choix technologiques qui ont été faits pour arriver à ce but ont permis de l'utiliser en tant que Web Service pour dispositif. Cinq fonctionnalités sont supportées par UPnP :

- **Recherche et découverte** : les dispositifs (les fournisseurs de services, appelés serveurs ou devices en UPnP) sont découvrables sur réseau local passivement ou activement, grâce à des messages diffusés en UDP.
- **Description** : les serveurs peuvent fournir un ou plusieurs services, qui implémentent des méthodes et fournissent plusieurs sources d'événements par des variables événementielles (*evented variables*). Ces capacités sont fixées et décrites dans des fichiers de description des serveurs et services UPnP, au format spécifique à UPnP, le SCPD (*Service Control Protocol Description*).
- **Invocations synchrones** : les appels de méthode par SOAP sont identiques aux appels de méthode des Web Services classiques.
- **Souscription aux événements** : s'abonner aux variables événementielles permet de recevoir des notifications de changement de leur valeur.
- **Présentation** : en plus des fonctionnalités de Web Service pour dispositif, UPnP permet la présentation de pages html sur un serveur HTTP embarqué. Cette fonctionnalité d'UPnP est la plus utilisée par les industriels, qui s'en servent essentiellement pour fournir une page web de configuration du dispositif ou de téléchargement de pilote pour utiliser le dispositif UPnP avec des moyens plus conventionnels que les appels de méthodes. UPnP remplit donc toutes les conditions pour être classé parmi les Web Services pour Dispositifs. Cependant, l'utilisation de fichiers de descriptions et d'événements de syntaxe spéciale, et l'absence de gestion de sécurité telle que l'authentification ou le cryptage révèlent certaines lacunes dans cette première implémentation, résolus dans UPnP version 2 nommé DPWS.

DPWS *Device Profile for Web Service* [13], définit un ensemble minimal de spécifications pour la recherche, la découverte, la description, la transmission de messages, et la diffusion d'événements liés à un dispositif [12]. DPWS est en ce sens similaire à l'approche UPnP,

dont il est en fait la deuxième version, mais est entièrement compatible avec les standards des Web Services en incluant de nombreuses extensions basées sur WSDL et SOAP pour l'intégration de services pour dispositifs dans des approches SOA. Dans ces extensions, on retrouve :

- WS-MetadataExchange qui définit le format des fichiers de description des services. En plus des informations présentes dans les descriptions des dispositifs et services UPnP, apparaissent les traductions de certaines informations dans plusieurs langues, et diverses URL pointant vers les adresses de retour des invocations, de présentation, d'accès aux autres fonctionnalités, etc.
- WS-Discovery qui définit les normes de recherche et découverte. En plus de se baser sur WSDL et SOAP, les améliorations par rapport à UPnP sont la faculté d'utiliser de façon transparente un annuaire local de services et l'ajout de critères de recherche supplémentaires, avec l'utilisation d'une syntaxe LDAP qui supporte des filtres comportant plusieurs conditions logiques.
- WS-Eventing qui définit les normes pour la souscription et l'envoi d'événements. L'envoi d'un événement peut se faire suivant plusieurs protocoles, le plus classique étant le push, semblable aux évènements d'UPnP. Le fournisseur d'événements fournit une fonctionnalité de filtre des messages envoyés, qui peut être configuré lors de la souscription par le consommateur. L'envoi d'un événement n'oblige plus à envoyer les valeurs de toutes les variables d'un service, comme c'était le cas avec UPnP. Il est aussi possible de déporter la gestion des souscriptions, hors de l'entité qui génère les événements.
- WS-Security qui définit les normes d'authentification, de signature et de cryptage des données transmises par le serveur. C'est un apport important par rapport à UPnP, qui permettra d'utiliser des services Web pour dispositifs pour lesquels l'utilisation par une personne mal intentionnée serait critique.

3.4 Conclusion

Pour adapter les applications à des conditions d'utilisation variables, les solutions industrielles actuelles de programmation par composants ne conviennent pas. Par conséquent, la conception et la construction de plateformes permettant l'adaptation des applications à leur contexte est aujourd'hui un thème majeur de recherche. Avant de présenter la plateforme WComp et les exemples d'applications et d'adaptations, il convient de passer en revue les divers *middlewares* (ou intergiciels) existants dans ce domaine.

4 Les plateformes pour l'adaptation au contexte

4.1 Tour d'horizon

Cette section du rapport présentent succintement les principaux intergiciels existants dans le domaine de l'adaptation des applications à leur contexte. Ces intergiciels diffèrent par leur manière de traiter les informations liées au contexte, mais aussi par leurs mécanismes d'adaptation.

4.1.1 Gaia

Motivation
L'intergiciel Gaia [14] offre la possibilité de développer des applications sensibles au contexte centrées sur l'utilisateur. Ces applications 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 distribué 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.

Principe
Gaia se base sur la notion d' "*active spaces*" [15] (environnements ubiquitaires), qui est un espace physique administré par une entité logicielle sensible au contexte permettant la gestion de dispositifs hétérogènes. Les applications sur Gaia sont à base de composants et distribuées, la gestion de ces composants pouvant se faire de manière distante. La forte dynamicité de l'environnement, et par conséquent la variabilité du contexte, ont impliqué l'utilisation des événements afin d'être réactif à ces variations. Cette tâche est accomplie grâce à l' "*event manager*". Ce *middleware* étant "*resources-aware*", il est nécessaire de posséder des informations relatives aux entités présentes dans un *active space* (que ce soit des applications, utilisateurs, dispositifs ou encore des services). Ainsi le "*physical entity presence subsystem*" détecte les nouvelles entités et ces dernières utilisent par la suite un système de *heartbeat* afin de confirmer leur existence.

Prise en compte du contexte
Le contexte est pris en compte grâce au "*context service*" qui se compose de "*context providers*". Ceux-ci fournissent aux applications des informations relatives au contexte, obtenues par différents capteurs. D'autre part ce service comprend également des composants capables de déduire des informations de plus haut niveau à partir de celles obtenues par les *providers*. Enfin un annuaire indique les *providers* disponibles.

4.1.2 RSCM

Motivation
RCSM pour "*reconfigurable context-sensitive middleware*" [16] a pour objectif de faciliter la réalisation d'applications sensibles au contexte dans un environnement ubiquitaire. Le développeur ne se préoccupe plus des aspects d'analyse du contexte ni de l'obtention des informations qui y sont relatives. De même il ne doit plus se préoccuper de l'adaptation au contexte.

Principe
RSCM fournit 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 composent de la dite interface sensible au contexte en plus de l'implémentation de l'objet qui en est indépendante. Le middleware fournit également un "*context analyser*" et un "*context sensitive service*" pour découvrir de nouvelles entités.

Prise en compte du contexte
RCSM utilise une structure permettant de collecter des informations relatives au contexte mais n'interprète pas ces données, il s'agit d'informations de bas niveau. Par la suite les composants peuvent s'inscrire auprès des collecteurs souhaités en fonction de leurs spécifications. L'utilisation d'interfaces sensibles au contexte permet de séparer l'implémentation fonctionnelle de la gestion du contexte.

4.1.3 CARMEN

Motivation
CARMEN signifie "*Context Aware Ressource Management Environment* ". Ce middleware a pour objectif de gérer les ressources de réseaux sans fils avec les problèmes de connexions intrinsèques à cette technologie. [17]

Principe
CARMEN utilise des proxys 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.

Prise en compte du contexte
Des métadonnées sont utilisées pour représenter les caractéristiques du contexte ; elles peuvent décrire les ressources composant le système, leur organisation et fournir des spécifications pour les opérations de gestion. On trouve dans l'architecture de CARMEN deux entités principales que sont le "*metadata manager*" et le "*context manager*". Le premier permet l'écriture, la mise à jour, l'évaluation des métadonnées. Le second détermine dynamiquement le contexte d'un client de CARMEN. Ce *middleware* suit le fonctionnement consistant dans un premier temps à obtenir des informations relatives au contexte (localisation, services disponibles...), par la suite l' "*event manager*" redirige les événements issus des variations de contexte vers les bonnes règles dans le "*metadata manager*". Ces informations remontent également directement vers le "*context manager*", qui évaluera le contexte pour présenter au client les références vers les ressources accessibles que le proxy du client contactera.

4.1.4 SOCAM

Motivation
SOCAM (*Service-oriented context aware middleware*) propose une architecture pour créer et prototyper des services sensibles au contexte. La conception de SOCAM repose sur l'idée de modéliser le contexte en utilisant des ontologies. [18]

Principe
L'architecture de ce *middleware* se compose d'un ensemble de services qui permettent à la fois d'obtenir des informations sur le contexte mais aussi de les interpréter. Cette architecture suit un modèle en couche :

- *Context sensing*
- *Context middleware*
- *Context application*

La première couche se compose de l'ensemble des capteurs permettant de remonter des informations relatives au contexte. Dans la couche *middleware* on retrouve les "*context providers*" qui récupèrent des informations des capteurs et les transforment en OWL. Ensuite le "*context interpreter*" raisonne sur ces informations. Les déclenchements de nouvelles actions ou les adaptations ont donc lieu grâce à un raisonnement sur une ontologie ou par des règles définies par l'utilisateur. Ces règles peuvent changer en cours d'exécution. Leurs validités sont vérifiées lorsque qu'un nouvel événement est propagé lors d'un changement de contexte.

Prise en compte du contexte
SOCAM utilise un modèle du contexte commun à tous les dispositifs sous la forme d'une ontologie. Celle-ci se découpe en deux 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, par exemple. Ce découpage offre la possibilité de réduire la taille de la connaissance du contexte. L'ontologie de bas niveau doit donc être mise à jour dynamiquement lors des changements de contexte. Un changement de domaine est défini soit par localisation, soit par l'utilisateur directement.

4.1.5 Amigo

Motivation
Amigo est un *middleware open-source* orienté service. Il a pour but de proposer un mécanisme pour intégrer dynamiquement des systèmes hétérogènes et de les faire communiquer entre eux. De nombreux protocoles existent et sont intégrés dans le middleware pour la découverte automatique de nouveaux services, laissant ainsi au développeur le libre choix du protocole à utiliser.

Principe
Le *middleware* peut se découper en trois catégories que sont le "*base middleware*", l' "*Intelligent User Services*" et le "*Programming and Deployment Framework*". La première fournit des moyens de communiquer et de découvrir les services présents dans le réseau, y compris des standards comme UPnP, *Web services* ... La seconde traite de ce qui est relatif au contexte, acquisition, interprétation, profils... Enfin le *Framework* fournit le nécessaire au développement de services "*amigo-aware* ".

Prise en compte du contexte
La prise en compte du contexte est donc gérée par l' "*Intelligent User Services*" qui se décompose lui-même en plusieurs composants :

- *Context management* : il récupère les informations relatives au contexte à partir des informations fournies par les capteurs, mais aussi en observant les applications en fonctionnement et les activités des utilisateurs. Il peut combiner ces informations pour ensuite les transmettre.
- *User modelling and profile management* : représentation du contexte et mise à jour pour raisonner sur le comportement futur de l'utilisateur, adapter les interfaces au contexte, aider l'utilisateur à obtenir des informations pertinentes.

- *Awareness and notification* : observe les variations de contexte et averti les applications de ces changements via des événements. Les applications n'ont ainsi plus à se préoccuper de la gestion des informations contextuelles, elles doivent juste définir des règles définissant quelles informations les intéressent. Ces règles sont posées en suivant la forme : événement-conditions-actions (ECA), elles peuvent intégrer des opérateurs logiques et relationnels. De même les utilisateurs peuvent créer leurs profils qui définissent comment et quand ils souhaitent être avertis (toujours à travers ces mêmes règles). Ainsi *l'awareness and notification service*, avant d'envoyer une notification regarde le profil utilisateur associé. Pour obtenir des informations concernant le contexte, il contacte le *context management service*.

4.1.6 CORTEX

Motivation
CORTEX pour "*CO-opérating Real-time senTient objects*" est un intergiciel ayant pour but d'aider à la conception d'applications sensibles au contexte. Ainsi il fournit des mécanismes pour la gestion du contexte et son interprétation. Ce middleware est implémenté sur openCOM, lui-même un intergiciel basé sur la technologie COM de Microsoft.

Principe
L'architecture de CORTEX se base sur un ensemble de quatre structures de développement de composants :
- *Publish/subscribe*
- *Service Discovery*
- *Context*
- *Resource Management*

La structure "*publish/subscribe*" se base sur un modèle d'événements et fournit des fonctionnalités comme le filtrage d'événements en fonction de leurs sujets, contenus ou en fonction du contexte. Le "*service discovery Framework*" permet de découvrir de nouvelles entités utilisant divers protocoles (UPnP, SLP...), il résout ainsi le problème de l'hétérogénéité des entités en jeu et propose sa propre représentation d'interfaces des services qu'il fournit. La structure "*Context*" apporte les mécanismes de gestion mais aussi de collecte d'informations relatives au contexte, il se base sur le modèle des objets sensibles ("*sentient object*"). Enfin la structure "Resource Management" offre un mécanisme de gestion des différentes ressources disponibles. [19]

Prise en compte du contexte
Comme évoqué précédemment la structure permettant la gestion du contexte se base sur le modèle des objets sensibles. Ces objets reçoivent des événements en provenance de capteurs ou d'autres objets sensibles et peuvent également produire des événements pour d'autres objets sensibles ou des déclencheurs ("*actuator*"). Ces derniers reçoivent des événements logiciels et ont pour but de les répercuter en matériel (dans le monde réel). L'objet lui-même se compose de trois entités. La première permet l'obtention d'informations relatives au contexte, la seconde modélise ces informations tandis que la dernière raisonne sur ces informations grâce à un moteur d'inférence. Ainsi une application dans CORTEX se compose d'un ensemble d'objets sensibles et de déclencheurs qui ont pour but d'adapter l'application au contexte via les événements reçus des objets sensibles. Le moteur d'inférence se compose d'un ensemble de règles, une règle étant spécifique à un objet sensible et devant permettre de répondre à un certain problème.

4.1.7 Aura

Motivation
Aura est un middleware orienté tâches qui se base sur la notion de "*personal aura*". Une "*personal aura*" joue le rôle de proxy pour l'utilisateur qu'elle représente, et se charge d'essayer d'obtenir les bonnes ressources pour la tâche que l'utilisateur souhaite réaliser. Pour cela une aura collecte des informations sur le contexte de l'utilisateur. Chaque tâche peut nécessiter différentes sources d'informations et applications. Contrairement à la plupart des autres intergiciels, Aura ne cherche pas à aider au développement d'applications sensibles au contexte mais cherche à utiliser celles existantes.

Principe
L'architecture d'Aura [20] se compose de quatre entités que sont le "*task manager*", le "*context observer*", l' "*environment manager*" et enfin les "*suppliers*". Ces derniers fournissent les services qui composent les tâches (édition texte, lecture vidéo...). Le "*task manager*" qui incarne la notion d'aura, permet de gérer quatre types de variations que sont le déplacement de l'utilisateur, les changements dans l'environnement, le changement de tâche d'un utilisateur et enfin les changements du contexte c'est-à-dire que la description d'une tâche se compose d'un ensemble de contraintes qui doivent être respectées. Lors d'un déplacement de l'utilisateur, il s'agit de déplacer également les informations concernant la tâche de l'utilisateur. Ainsi il devient possible de continuer la tâche commencée à un endroit dans un autre. Lors de changements dans l'environnement il devient nécessaire de vérifier que la qualité de service offerte est suffisante pour le bon déroulement de la tâche. Ainsi l'objectif est d'offrir à l'utilisateur un environnement cohérent pour la tâche qu'il souhaite réaliser. Le "*context observer*" collecte des informations sur le contexte et propage des événements lors de variations jusqu'au "*task manager*". Enfin l' "*environnement manager*" joue le rôle d'annuaire des "*suppliers*".

Prise en compte du contexte
Les informations contextuelles remontées par Aura concernent les utilisateurs, les dispositifs, l'espace physique et les réseaux. Ces informations sont contenues et fournies par l'environnement, la gestion de tout ceci se trouvant dans l' "*environment manager*". Un environnement est défini comme l'ensemble des capacités (services) offerts en un certain lieu, il ne s'agit pas des services qu'il est possible d'atteindre mais de l'ensemble de ceux qui offre une bonne utilisabilité [21]. Ainsi chaque lieu doit posséder son propre "*environment manager*" ayant la connaissance des ressources disponibles. De ces données le "*context observer*" cherche à définir l'intention de l'utilisateur, mais aussi sa tâche courante ainsi que sa localisation. Ainsi, si un utilisateur travaille dans son bureau et a une réunion à 10 heures, s'il quitte son bureau vers cette heure alors le système en déduit qu'il va à sa réunion. On voit donc ici l'importance d'informations comme la tâche courante ou encore la localisation. Le "*task manager*" contient également les préférences utilisateurs sous forme de règles qui sont évalués afin de déterminer de quelle manière le service va être rendu. Par exemple lorsque la bande passante est trop faible l'utilisateur préfère regarder sa vidéo plus tard.

4.1.8 CAMidO

Motivation
CAMidO (*Context-aware middleware based on ontology meta-model*) est un intergiciel qui a pour but de faciliter le développement d'applications sensibles au contexte. Il permet de séparer les mécanismes d'adaptation au contexte, pris en compte par le middleware, des mécanismes métier.

Principe
Le contexte est défini ici comme un ensemble de contextes observables. De ces derniers sont tirés les contextes collectés, eux-mêmes pouvant subir un traitement pour donner des contextes interprétés. Les contextes collectés et interprétés sont analysés et peuvent entrainer des décisions d'adaptation. Chaque déclenchement d'adaptation est conditionné par la présence d'un ou plusieurs contextes pertinents, un contexte pertinent étant représenté par un état ou une composition d'états des contextes observés. Ainsi, avec l'objectif de modifier l'application, le *middleware* suit le cheminement suivant : récupérer les informations concernant le contexte, interpréter ces informations, analyser ces données et prendre une décision concernant l'adaptation envisageable, et enfin, appliquer ces décisions.

CAMidO est donc un intergiciel orienté composant qui possède des entités capables de gérer le contexte et de réaliser des adaptations. Pour ce faire, son architecture repose sur cinq composants :

- *Collection manager* : s'occupe de récupérer les informations concernant le contexte.
- *Context analyser* : filtre les informations concernant le contexte et note les variations.
- *Context interpreter* : déduit de ces informations des "contextes interprétés".
- *Context repository* : base contenant les informations relatives au contexte.
- *Component adapter* : adapte l'application aux changements de contexte (en fonction de règles, d'une "politique").

Prise en compte du contexte
L'adaptation se fait grâce au compilateur de CAMidO qui transforme d'une part les informations fournies par un développeur en règles de logique de premier ordre qui serviront à l'interprétation des données relatives au contexte, et d'autre part le code source de l'adaptation qui est ensuite intégré au conteneur sous la forme d'un contrôleur pour automatiser l'adaptation. CAMidO considère deux types d'adaptations : les réactives, qui surviennent suite à un changement de contexte, et les proactives qui anticipent les variations et qui modifient les règles définissant le comportement d'un service. [22]

4.1.9 CARISMA

Motivation
Cet intergiciel a pour but de fournir des mécanismes permettant la prise en compte des changements de contexte en définissant des politiques de réaction. Les règles définissant ces politiques peuvent être conflictuelles, le *middleware* a également pour rôle de résoudre ces conflits. Ainsi CARISMA ("*Context Aware Reflexive Midleware for Mobile Applications*") utilise la réflexivité pour permettre aux développeurs de concevoir des applications sensibles au contexte grâce à ces mécanismes. [23]

Principe
Dans l'optique de permettre au *middleware* de modifier son comportement en fonction du contexte à l'exécution, CARISMA utilise la réflexivité, associée à des métadonnées. Celles-ci permettent d'obtenir une séparation entre ce que l'intergiciel fait et de quelle manière. Par contexte on entend ici tout ce qui peut avoir une influence sur le comportement d'une application. Les informations de ce type sont regroupées sous deux catégories : celles concernant les dispositifs (mémoire, batterie) et celles relatives à l'environnement (tout ce qui ne concerne pas les dispositifs). L'architecture de CARISMA se découpe en 4 composants :

- Le "*Core*" qui fournit les fonctionnalités basiques (communications...),
- Le "*Context Manager*" qui gère les différents dispositifs permettant d'obtenir des informations relatives au contexte, l'intergiciel ne collectant que des informations de bas niveau concernant le contexte et ne possédant pas de mécanisme d'interprétation,
- Le "*Core Services*" qui gère les informations relatives au contexte,

- L' "*Application Model*" qui aide à la création des applications avec une API pour la réflexivité.

Prise en compte du contexte
L'intergiciel CARISMA permet des adaptations au contexte grâce à l'ensemble de services configurables qu'il fournit. Les configurations se font à partir du profil des applications et grâce à la réflexivité. Lors de changements de contexte, l'intergiciel va consulter les informations contenues dans les métadonnées (profils utilisateur et application). De cette manière les applications peuvent "demander" au *middleware* d'avoir certaines réactions. Par exemple, dans le profil d'une application, il peut être indiqué que lorsque la bande passante est trop faible elle doit être déconnectée. D'autre part une application peut demander à CARISMA un service (par exemple accéder a certaines données) et à chacune de ces requêtes l'application doit fournir des règles, à base de logique du premier ordre, indiquant de quelle manière le *middleware* doit rendre ce service. Si l'intergiciel s'occupe de la modélisation et de la bonne cohérence des informations relatives au contexte, il est important de noter que plusieurs adaptations peuvent s'appliquer et CARISMA doit par conséquent gérer des conflits.

4.1.10 Oxygen

Motivation
Cet intergiciel a pour objectif de faciliter les communications entre les utilisateurs et les machines. Ainsi il devient possible de communiquer avec *Oxygen* grâce à de la reconnaissance vocale et des technologies de la vision. Il permet à un utilisateur de continuer à utiliser ses services même en déplacement.

Principe
Le système *Oxygen* se décompose en trois parties que sont H21, un service d'appareils 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 configurer à distance le H21 pour qu'il se mette dans un de ces modes en particulier. Les donnés des utilisateurs sont stockées dans un système de fichier particulier : SFS (*Self-certifying File System*), grâce auquel chacun gère ses droits et peut où qu'il soit accéder à ses données. Pour connaitre l'identité d'une entité localisée et pouvoir se ramener à des identifiants comme une URL ou une adresse IP, Oxygen possède un annuaire contenant les dispositifs présents et des attributs associés.

Prise en compte du contexte
Oxygen étant centré utilisateur on peut alors percevoir le cycle de son adaptation comme, dans un premier temps, une action de l'utilisateur qui entraine une capture des informations relatives au contexte dans l'environnement E21 puis leur évaluation pour enfin obtenir une adaptation. Dans *Oxygen* les communications entre les entités se font a l'aide de messages, les réactions sont obtenues à l'aide de règles action-conséquences. On évalue une action de l'utilisateur pour en tirer les conséquences. *Oxygen* est écrit au dessus de java qui lui offre ainsi la possibilité de charger ou détruire dynamiquement des objets. Les adaptations sont donc des chargements et des destructions d'objets mais aussi des modifications de leurs abonnements aux messages. Il s'agit donc de faire évoluer un graphe.

4.1.11 SAFRAN

Motivation
Le middleware SAFRAN – Self-Adaptative FRActal compoNents – se base au dessus de Fractal, il utilise une approche par aspects pour rendre modulaire le code d'adaptation d'applications mais aussi afin d'utiliser la dynamicité offerte par le tissage (dynamique). 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 sur l'adaptation de la structure des applications.

Principe
Afin de réaliser des adaptations SAFRAN utilise une approche par aspects qui offre une séparation du code de l'adaptation du code métier et permet grâce au mécanisme du tissage des aspects de les appliquer et de les retirer dynamiquement. [24]
SAFRAN repose sur le principe que l'adaptation d'une application correspond au cycle :
1. Collecte des informations relatives au contexte
2. Prise de décision des modifications à réaliser
3. Application de ces modifications

Comme évoqué précédemment, SAFRAN repose sur Fractal qui permet de réaliser des applications par assemblages de composants. Un composant se découpe en un contrôleur et un contenu. Le contrôleur permet via des interfaces de reconfigurer le composant ou encore de diriger des messages vers la bonne destination. Le contenu peut regrouper d'autres composants, on parle alors de composants composites. Les scripts, écrits à l'aide de *Fscript*, permettent d'utiliser ces mécanismes pour réaliser des adaptations.

Prise en compte du contexte
Fscript offre la possibilité d'exprimer des règles d'adaptation. Ce langage permet aussi la manipulation de variables et offre une logique du second ordre. D'autre part *Fscript* garantit qu'après une modification, l'application sera toujours utilisable. Les points de jonction sont évalués avec l'apparition d'événements. On dénote deux types d'événements, les endogènes liés à l'exécution de l'application et les exogènes liés aux changements dans l'environnement. Ces derniers sont obtenus grâce à un *framework* appelé *WildCAT*, qui collecte les informations relatives au contexte. Celui-ci modélise le contexte dans des domaines contextuels qui sont eux-mêmes modélisés sous une forme arborescente. Si *WildCAT* propose un ensemble de sondes, il en facilite également la conception ; il est de plus possible de spécifier des règles permettant de déclarer sous quelles conditions *WildCAT* doit propager un événement (ex : une luminosité minimum). D'autre part il est possible d'indiquer à une sonde quand elle doit effectuer ses mesures. Les règles d'adaptation suivent la syntaxe when <event > if <condition> do <action> soit le format ECA. L' "*adaptation controller*" associera ces règles aux composants et aux événements. Ainsi la réception d'un événement déclenche une adaptation si les conditions sont vérifiées. On peut donc considérer que les événements exogènes sont un nouveau type de point de jonction.

4.2 Les spécificités de chacun

Tout d'abord, il convient de situer les centres de recherches étudiant l'adaptation des applications à leurs contextes. Voir aussi en annexe 1 la liste des centres de recherches utilisant des intergiciels sensibles au contexte.

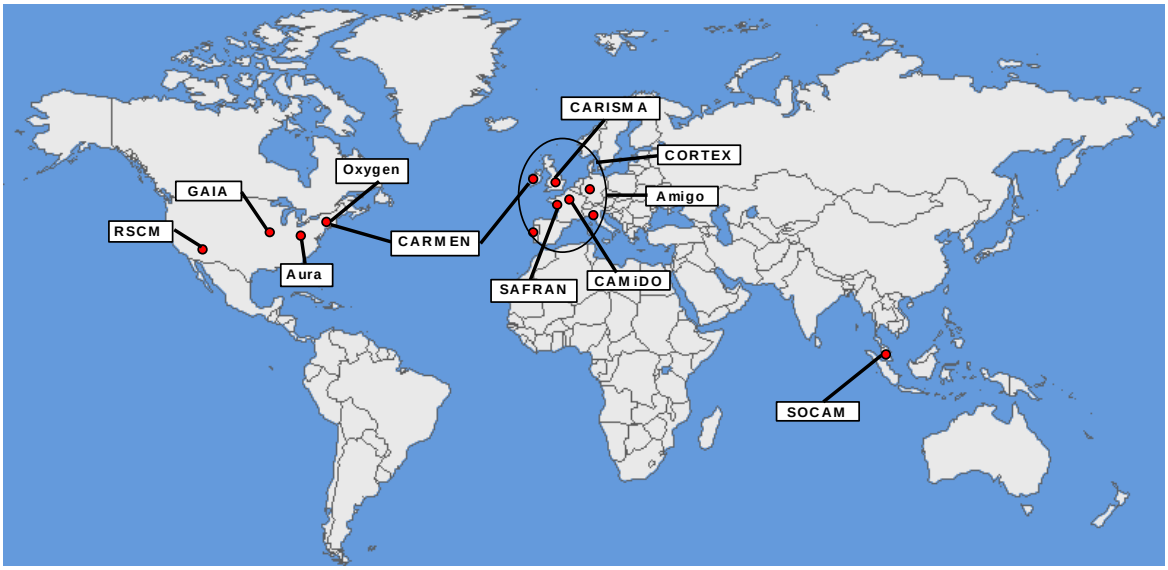


Figure 11 : **Les intergiciels sensibles au contexte dans le monde**

Chacun de ces intergiciels cible un problème particulier lié à l'adaptation des applications au contexte. Par exemple, Aura et GAIA sont orientés tâches et adressent des problèmes dus à la mobilité de l'utilisateur, CORTEX se soucie des communications et de la qualité de service entre tous les objets d'un environnement d'informatique ambiante, et *Oxygen* vise à améliorer les interactions entre les utilisateurs et dispositifs mobiles. [9]

Tous ces intergiciels visent à faciliter la conception d'applications en informatique ambiante. Les plateformes logicielles sont dynamiques et fournissent des capacités d'adaptation pour les applications qu'elles exécutent. Ces différentes approches utilisent des paradigmes de programmation qui ont été abordés au chapitre 2.5.1 du présent rapport. Parmi ces paradigmes, tous ne présentent pas les mêmes avantages. En effet, la conception orientée composants est plus pertinente pour l'adaptation que l'architecture orientée service, cette dernière étant particulièrement adaptée à la gestion de l'hétérogénéité et à la distribution des entités. Par conséquent, il est nécessaire de faire émerger des systèmes multi-paradigmes, comme les Web Services pour dispositifs, qui intègrent les concepts des services et des plateformes à évènements, ainsi que la découverte dynamique distribuée. Alors qu'ils existent depuis plusieurs années (1999 pour UPnP, 2004 pour DPWS), une convergence est en train de se produire avec l'arrivée des nouvelles architectures orientées services, avec le nom controversé SOA 2.0, qui intègre les notions des plateformes à évènements aux SOA.

Un exemple de ces combinaisons de paradigmes est SCA (Service Component Architecture). SCA définit une architecture de composants et de services, utilisant les composants pour manipuler des orchestrations de services et créer des services composites. SCA correspond bien à la création d'applications dans les systèmes d'information, mais n'est pas adapté aux cibles légères, et à la dynamicité des environnements pervasifs d'informatique ambiante. En effet, une plateforme d'exécution s'appuyant sur SCA embarque un certain nombre de propriétés non fonctionnelles aussi appelées services techniques de la plateforme, dans le but de faciliter la programmation des parties non fonctionnelles des composants, comme la gestion du cycle de vie, la persistance, ou la synchronisation des flots d'exécution. Or, plus une plateforme offre de services techniques, moins il est simple de contrôler l'exécution de l'application précisément, et d'adapter son comportement. [9]

C'est dans ce contexte que le modèle SLCA a été créé. Il a pour but de définir une architecture dynamique de composition de services, prenant en compte toutes les problématiques de l'informatique ambiante, en tirant avantage des divers paradigmes présentés.

5 SLCA : une approche multi paradigmes

5.1 Assemblage de composants légers

Le modèle de composants LCA (*Lightweight Component Architecture*) (voir Annexe 2) est un modèle de *Beans* [25] légèrement modifié, adapté à d'autres langages de programmation, avec les concepts de ports d'entrée et de sortie et de propriétés. Un composant est toujours une instance d'un type ; il possède un nom unique et une interface composée de deux ensembles d'´evènements et de méthodes, mais n'est pas nécessairement sérialisable. Les types des composants définissent les interfaces des instances.

Soient T la liste des types de composants chargés dans le container, C l'ensemble des instances de composants, E l'ensemble des évènements, et M l'ensemble des méthodes. Les déclarations des évènements et des méthodes sont regroupées dans le terme "port", dont une liste forme l'interface d'un composant. On considère un ensemble de liaisons L qui permettent de passer le flot de contrôle d'un composant à un ou plusieurs autres. Elles sont définies par un port de sortie d'une instance, soit un élément de E, et par un ou plusieurs ports d'entrée d'une ou plusieurs instances, membres de M. Un sous-assemblage de composants est une union d'une partie de C et d'une partie de L. Un assemblage est un sous-assemblage pour lequel aucune liaison n'est pas reliée sur un évènement et au moins une méthode.

Le modèle est totalement dynamique. Les chargements et déchargements de types de composants (modifications de T), instanciations et destructions d'instances (modifications de C) et création et destruction de liaisons (modifications de L) entre les instances sont possibles pendant l'exécution de la plateforme. Les évènements – aussi appelés mécanisme "push", ou inversion de contrôle – sont utilisés dans les containers légers et cet usage n'est pas limité à l'augmentation de la réactivité : ils donnent une plus grande dynamicité aux composants, principalement en donnant des points d'entrée/sortie explicites permettant de les relier ou délier pendant l'exécution. Aucun code n'a besoin d'être généré, ni étudié pour trouver à quel endroit modifier un composant pour changer la cible de sa liaison.

5.2 Le modèle Bean4WComp

Le modèle Bean4WComp [27] est un modèle de composant LCA. Chaque composant émet des événements pour informer les autres d'un changement d'état interne ou pour invoquer une méthode d'un autre composant. Les composants réagissent à la réception d'un événement, par exemple en émettant eux aussi des événements.

5.3 Le modèle SLCA

SLCA (*Service Lightweight Component Architecture*) est un modèle d'architecture qui utilise des composants légers pour concevoir des *Web Services* composites. Un service composite encapsule le container SLCA qui contient un assemblage dynamique de composants légers.
Le modèle SLCA repose sur un environnement d'exécution logiciel et matériel évoluant dynamiquement. Cet environnement est défini par un ensemble de ressources, comme autant d'entités logicielles ou physiques dont l'apparition et la disparition ne sont pas pilotées mais en général subies par l'application.

Cette architecture prend en compte deux paradigmes principaux :

- ***Web services* basés sur des évènements**. Ce sont des *Web Services* pour dispositifs comme UPnP ou DPWS. Les applications d'informatique ambiante sont alors un graphe de *Web Services* utilisant des évènements.
- **Assemblages de composants légers** dans les *Web Services* composites utilisant des évènements. Un service composite est basé sur un assemblage interne de composants pour gérer la composition entre d'autres *Web Services* utilisant des évènements et pour concevoir dynamiquement l'interface d'un nouveau service composite.

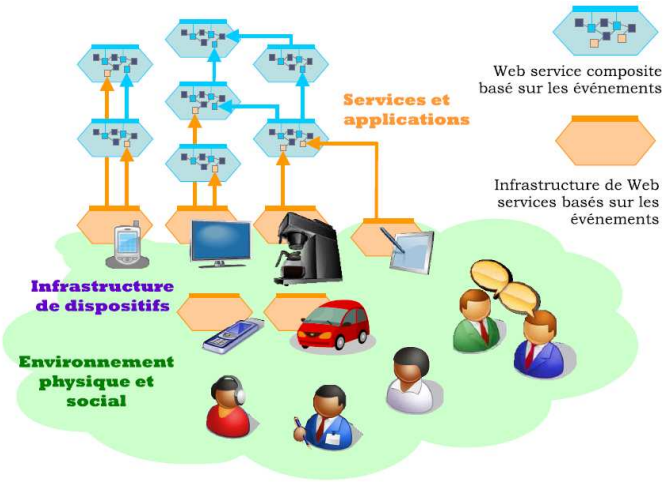


Figure 12 : **Graphe de Web Services basés sur les évènements**

SLCA définit donc un modèle d'architecture compositionnelle basé sur des évènements, pour concevoir des *Web Services* composites. [9]

5.3.1 Composition pour de nouveaux services

SLCA est basé sur une infrastructure de services utilisant des évènements, ou de *Web Services* pour dispositifs. Ces services permettent d'utiliser des dispositifs, ou d'autres programmes, dans les applications conçues avec SLCA. Les applications sont conçues par composition de services par assemblage de composants, et les nouvelles fonctionnalités ainsi créées peuvent être à leur tour fournies sous forme de services. Un assemblage de composants peut donc créer une application en elle même, ou un nouveau service composite qui sera utilisé dans d'autres applications. Un service composite (container) fournit deux interfaces de services (figure 13).

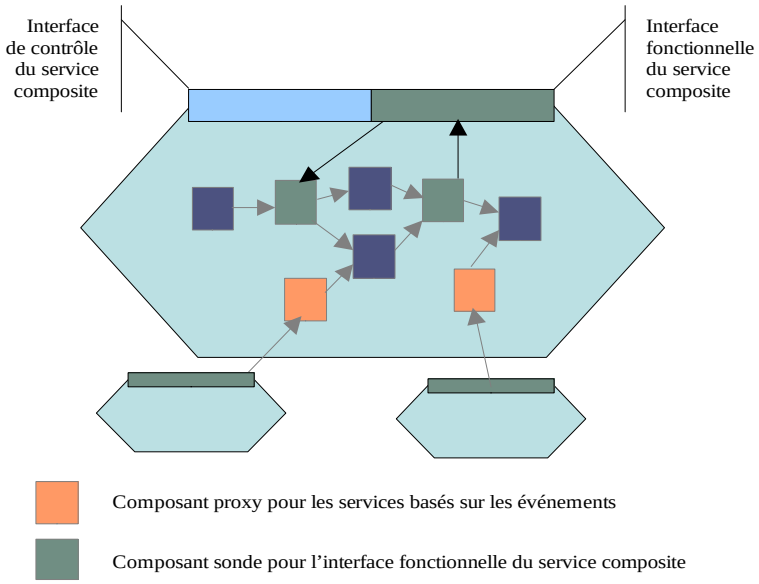


Figure 13 : **Web Service composite basé sur les évènements**

La première interface (interface de contrôle) permet de modifier dynamiquement l'assemblage interne de composants SLCA, la deuxième (interface fonctionnelle dynamique) permet de publier et accéder aux nouvelles fonctionnalités fournies par l'assemblage de composants en tant que Web Service composite.

L'interface de contrôle permet de modifier dynamiquement l'assemblage interne du service, en ajoutant ou en supprimant des composants et des liaisons. Ces modifications sont réalisées à partir d'un client connecté (il est possible que ce soit un autre service composite qui utilise un composant proxy sur ce service).

L'interface fonctionnelle dynamique exporte les évènements et les méthodes de l'assemblage interne en utilisant les composants sondes. L'ajout ou la suppression d'un composant sonde modifie dynamiquement l'interface fonctionnelle et la description du service composite correspondant (qui est un Web Service pour dispositif). Techniquement, deux composants sondes existent : le puits, qui ajoutent une méthode à l'interface du service composite, et qui dans l'assemblage de composants interne n'a qu'un port de sortie (figure 14). L'invocation d'une méthode sur le service composite émet donc un évènement dans l'assemblage de composants interne. Le deuxième composant sonde est la source (figure 15), qui ajoute un évènement à l'interface du service composite, et n'a qu'un port d'entrée. L'invocation d'une méthode dans l'assemblage de composants émettra un évènement de *Web service*. Sur les figures 14 et 15, l'interface de contrôle n'est pas représentée.

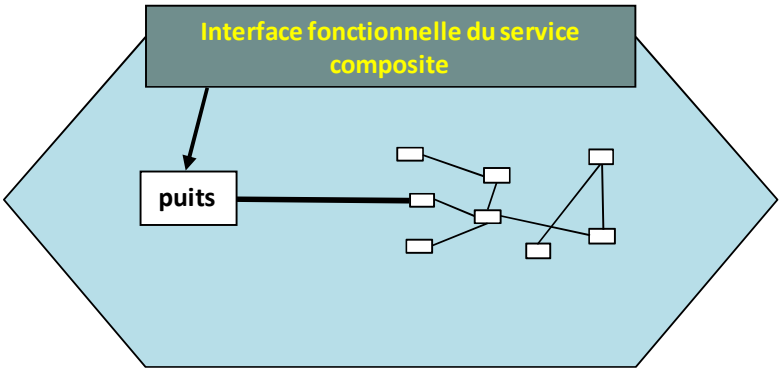


Figure 14 : **Les composants sondes : le puits**

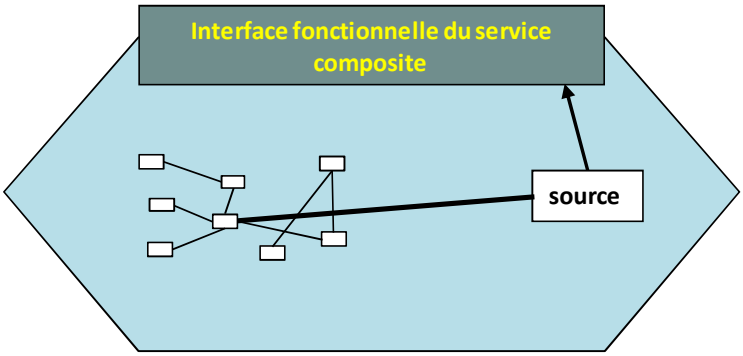


Figure 15 : **Les composants sondes : la source**

Parmi les composants légers que l'on instanciera dans un container (service composite) pour créer une application ou un service de plus haut niveau, certains peuvent être des composants proxys de Web Services, basés sur les évènements ou non. Ainsi, un composant composite peut contenir un composant communiquant avec l'interface fonctionnelle ou structurelle d'un autre composant composite. La notion de hiérarchie de composition est ainsi introduite. [9]

Voir en annexe le méta-modèle SLCA (annexe 3).

5.3.2 Architecture de SLCA

SLCA définit une approche multi-design du développement par composition. L'architecture de SLCA s'organise autour de *containers* et de *designer.* [26]

Les containers sont les entités qui contiennent les assemblages de composants légers, et qui possèdent les deux interfaces de services pour fournir des services composites. Leur objectif est de prendre en charge dynamiquement la gestion de comportements tels que l'instanciation, la désignation, la destruction de composants logiciels fonctionnels et de liaisons.

Les designers sont connectés à l'interface de contrôle des containers, agissant comme des consommateurs de services. Ils permettent de manipuler ou visualiser les assemblages de l'application en utilisant les formalismes adaptés. On peut par exemple citer deux *designers* : le *designer* visuel d'assemblage et le *designer* de *Web Services* pour dispositifs *UPnPProxyDesigner*.

Le designer graphique d'architecture permet de composer manuellement des assemblages de composants à partir d'une représentation graphique des flots d'évènements. L'application est construite en assemblant des composants sur étagères et en les reliant par diffusion d'évènements. Cette méthode de conception est avantageuse pour une conception rapide d'application mais devient limitée lorsque la fréquence d'apparition et de disparition augmente.

Le designer de composants proxys permet de gérer l'apparition ou la disparition des composants proxys de Web Services pour dispositifs dans le container. Le designer de composants mixtes UPnP génère et instancie automatiquement les composants proxys de dispositifs UPnP, en utilisant les descriptions des Web Services. Il gère les variations de l'infrastructure sans intervention directe de l'utilisateur. Il ne traite pas de la modification

d'assemblage de composants mais uniquement de leur instanciation/destruction. L'utilisateur ne peut agir sur le comportement de ce *designer*, et donc sur le service composite qui lui est associé, qu'en modifiant l'état réel des dispositifs physiques.

On dispose d'une plateforme qui intègre la totalité des concepts de SLCA. En effet, la plateforme SharpWComp 2.0 est une implémentation en C# du modèle de composants SLCA, qui utilise les *delegate* de .NET pour les évènements locaux, et UPnP pour les interfaces des *containers* et *designers* et les communications avec les dispositifs de l'environnement. La plateforme WComp fonctionne dans un environnement Microsoft .NET. Une implémentation en Java existe mais ses fonctionnalités sont encore limitées.

5.4 SharpWComp 2.0

L'environnement de programmation choisi pour supporter la plateforme WComp est SharpDevelop. Il s'agit d'un environnement de développement intégré (IDE) libre, gratuit et équivalent à Visual Studio pour l'environnement .NET. SharpWComp a été conçu comme une extension (plugin ou addin) à SharpDevelop. Ceci permet de bénéficier de l'ensemble des fonctionnalités de SharpDevelop, comme par exemple la génération de code correspondant à l'assemblage WComp, la manipulation de la représentation graphique de l'application et la génération du code exécutable correspondant à l'application. L'extension SharpWComp ayant été compilée avec la version 1.0.2a de SharpDevelop, il est impératif d'utiliser cette version, sous peine de ne pouvoir faire fonctionner SharpWComp.

WComp signifie à l'origine "Wearable COMPuter". Le projet est parti d'une idée originale d'étudiants et d'enseignants chercheurs de l'Université de Nice - Sophia Antipolis (UNS). Son développement a été, à l'origine, soutenu par l'Ecole Supérieure en Sciences Informatiques (ESSI) maintenant département Sciences Informatiques de l'Ecole Polytechnique de l'Université de Nice - Sophia Antipolis (Polytech' Nice - Sophia). Cette plateforme est un des logiciels développés au sein de l'équipe de recherche *Rainbow* du Laboratoire I3S (CNRS / Université de Nice – Sophia Antipolis).

WComp est au cœur d'un environnement expérimental original : une plateforme d'étude des usages des équipements informatiques mobiles en environnement simulé, appelée "Ubiquarium Informatique".

5.4.1 L'Ubiquarium[1]

L'Environnement expérimental de WComp, appelé Ubiquarium est constitué de divers dispositifs, comme autant de services découvrables et composables dynamiquement par WComp. Ces dispositifs peuvent être soit des dispositifs virtuels (objets d'une scène 3D dans laquelle l'utilisateur est immergé), soit des dispositifs réels portés par l'utilisateur ou présents dans son environnement. Tous les équipements de l'Ubiquarium, réels ou virtuels, sont donc basés sur une interface de type Web Service (classique ou pour dispositifs) ce qui limite ainsi au maximum les conceptions ad-hoc. L'Ubiquarium actuellement mis en oeuvre repose sur trois grandes classes d'équipements :

[1] du Latin "ubique", en toute chose et tout être, avec le suffixe "rium" signifiant lieu ou structure. Donc Ubiquarium Informatique : "lieu ou structure dans laquelle l'informatique est en toute chose et tout être".

- dans l'environnement réel de l'utilisateur : des dispositifs sans-fil présents dans l'environnement, tels que des capteurs (luminosité, température, accéléromètre, ...) et actionneurs (télé-relais, ...),
- sur l'utilisateur : des dispositifs d'interaction: joystick, téléphone portable, PDA, "Wearable Computer",
- dans l'environnement simulé : sous forme d'une scène virtuelle 3D, des dispositifs virtuels UPnP associés à des objets de la scène.

Un tel environnement est un cadre idéal pour l'évaluation de nouvelles applications de l'informatique mobile et ambiante telles que les usages des ordinateurs portés ou "*Wearable Computers*". Cette plateforme est donc tout particulièrement adaptée à l'étude des mécanismes d'adaptation logicielle pour des applications de l'informatique mobile et ubiquitaire sensibles au contexte.

5.4.2 La plateforme WComp

WComp est une plateforme à composants pour le développement rapide de prototypes d'applications multi-dispositifs qui doivent évoluer avec leur environnement d'exécution. La plateforme a pour vocation de gérer à la fois une application comme un assemblage de composants logiciels et la dépendance de certains composants par rapport aux ressources de l'infrastructure. Elle permet l'adaptation dynamique des applications en informatique ambiante reposant sur une infrastructure composée à la fois de Web Services et de Web Services pour dispositifs. Intel met à disposition un ensemble d'outils pour travailler avec l'infrastructure UPnP, dont certains seront utilisés par la suite.

La gestion et les modifications des assemblages de composants WComp peuvent utiliser différents modèles. L'architecture de WComp s'organise donc autour de containers et de designers [26]. L'objectif des containers est de prendre en charge dynamiquement et à l'exécution la gestion de comportements tels que l'instanciation, la désignation, la destruction de composants logiciels fonctionnels et de liaisons.

La plateforme WComp offre donc la possiblité de créer des *containers*, mais aussi des proxy pour *Web Services*, des proxy pour *Web Services* pour dispositifs et des modèles de code pour écrire des *bean* (voir figure 16, respectivement 4, 1, 2 et 3)

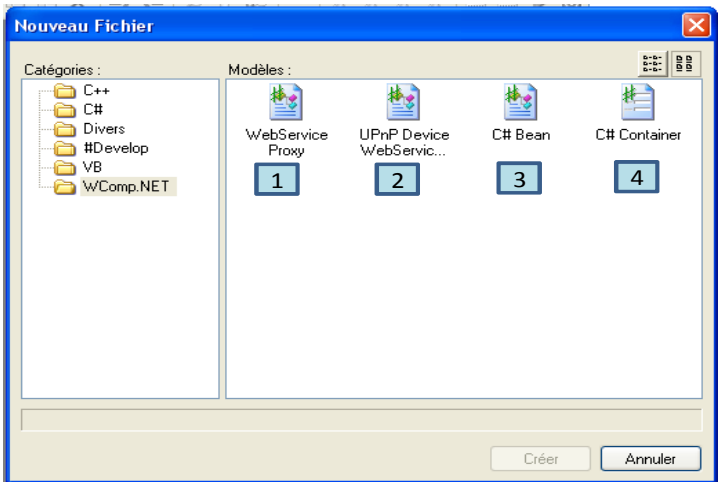


Figure 16 : **Création de fichiers avec WComp**

En ce qui concerne les *containers*, La plateforme offre un accès à plusieurs modes pour l'édition de l'application. La première vue qui apparaît est l'éditeur de texte du code source

de l'application : le concepteur Source (voir fig. 17). On dispose aussi du concepteur graphique SharpWComp (fig. 18) et du concepteur *Design* (fig. 17) permettant le placement des composants graphiques (widgets) de l'application. Les onglets en bas de la vue principale doivent être utilisés pour passer d'un concepteur à l'autre.

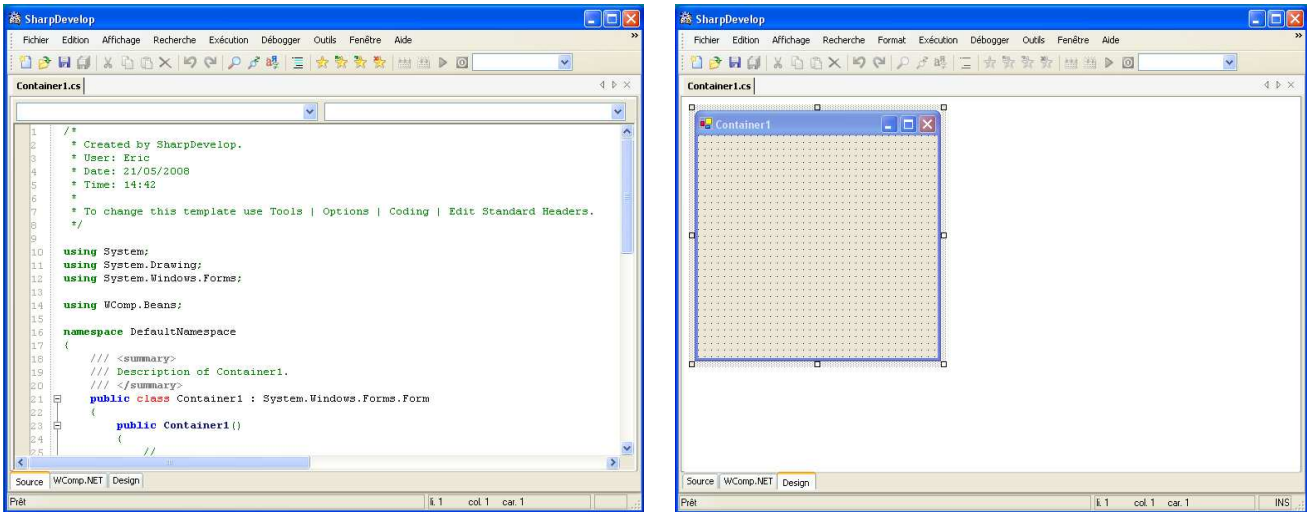


Figure 17 : **Concepteur Source et concepteur *Design***

La plateforme WComp gère des assemblages de **différents types de composants** logiciels. On en compte deux :

- Les composants mixtes : ce sont des composants logiciels présents dans la plateforme uniquement si les ressources qu'ils utilisent sont accessibles. Ils peuvent apparaître ou disparaître au gré de l'apparition et de la disparition des ressources dans l'infrastructure de la plateforme décrite précédemment. Il s'agit principalement de clients de services (c'est-à-dire les composants proxy).
- Les composants logiciels : ce sont les composants qui ne dépendent pas de la variation des ressources.

Les composants mixtes :

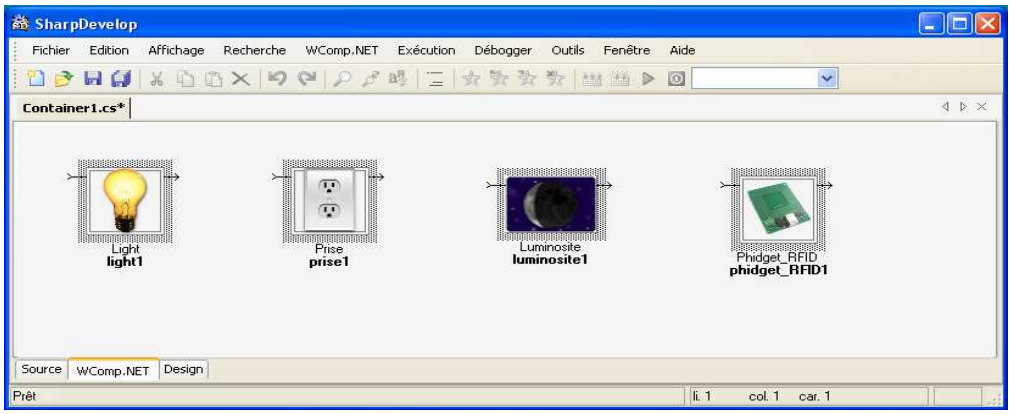


Figure 18 : **Quatre composants mixtes**

Sur la figure 18, on a un container WComp contenant quatre composants mixtes représentant trois ressources physiques et une logicielle. En effet, le composant "light1" est un composant proxy pour la lampe virtuelle mise à disposition dans les outils UPnP Intel, tandis que les trois autres composants représentent bien des dispositifs physiques. Le composant "prise1" est un proxy pour une prise RF pour laquelle a été développé un serveur UPnP. Le composant "luminosite1" est un proxy pour un capteur de luminosité. Le composant "phidget_RFID1", quant à lui n'est pas un proxy mais interagit directement avec un lecteur RFID connecté sur la machine.

Figure 19 : **Ressources matérielles et logicielles**

Parmi les composants mixtes, on compte également les proxys pour Web Services. La plateforme WComp permet de générer automatiquement des proxys pour Web Services et des proxys pour Web Services pour dispositifs (fig.16, respectivement icônes 1 et 2). La description du *web service* au format WSDL est utilisée pour générer un composant proxy WComp correspondant à un Web service (fig. 20)

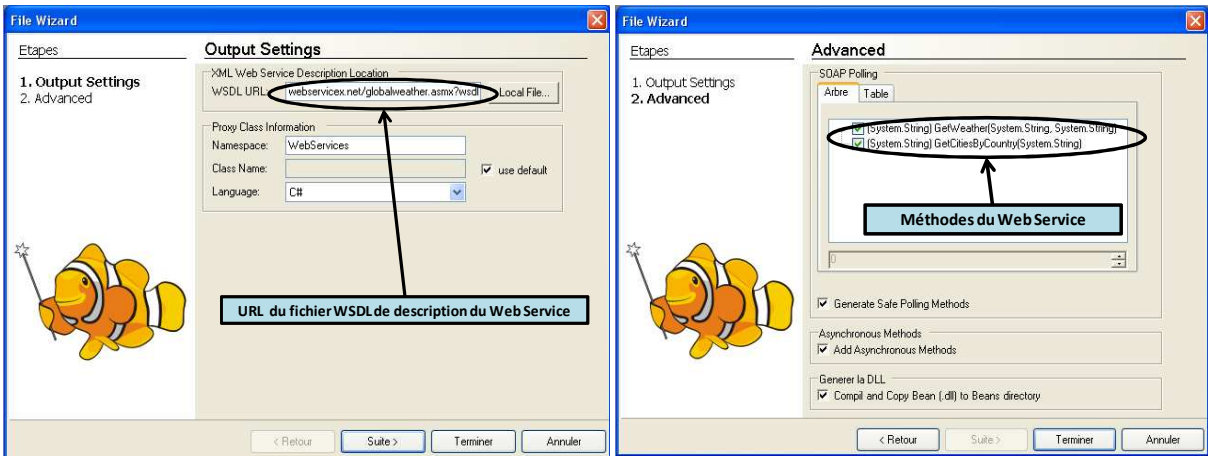


Figure 20 : **Création d'un proxy pour *Web Service***

Les méthodes mises à disposition par le web service apparaissent dans l'interface de création du proxy. Ici, il s'agit d'un Web Service fournissant la météo.

La plateforme WComp détecte automatiquement les dispositifs UPnP présents sur le réseau. Il suffit de sélectionner les dispositifs pour lesquels on désire créer un proxy. Dans le cas présent, seule la lampe virtuelle Intel (voir fig. 21) est disponible sur le réseau.

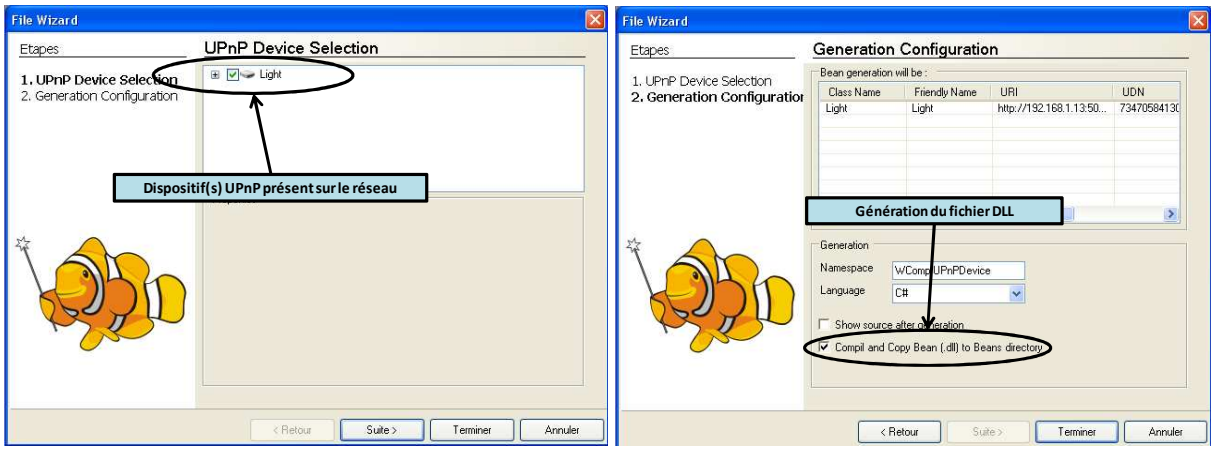


Figure 21 : **Création d'un proxy pour *Web Service* pour dispositif**

Les manipulations décrites sur les figures 20 et 21 permettent de créer des fichiers ".DLL" qui représentent les proxys des *Web Services*.

Les composants logiciels :
Un développeur a la possibilité de créer ses propres composants : les *beans*. Ils doivent suivre une syntaxe particulière et la plateforme WComp fournit un modèle de code pour ce type de composants. Les composants type "Windows Forms" sont aussi des composants logiciels. Enfin, on compte également parmi ces composants les composants "emitProbe" et "sourceProbe", qui permettent respectivement de créer des méthodes et des évènements pour un nouveau service composite.

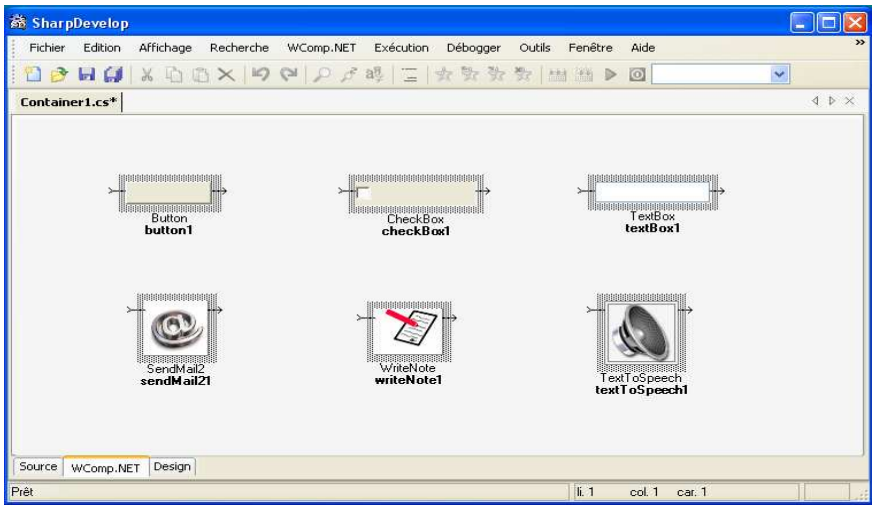


Figure 22 : **Quelques composants logiciels**

Le code d'un composant (*bean*) se décline en quatre parties :
- les propriétés du composant,
- les méthodes publiques, c'est à dire les méthodes qui pourront être invoquées sur le composant,
- les méthodes privées (éventuellement un constructeur),
- les évènements pouvant être émis par le composant.

Voir en annexe 4 le modèle de code fourni par la plateforme pour créer un *bean*.
Tous ces composants logiciels sont exécutables dans un environnement d'exécution donné si celui ci possède de manière permanente les ressources logicielles et matérielles pour l'exécuter. Dans le cas de l'Ubiquarium, les ressources sont limitées aux services et dispositifs accessibles. Les composants mixtes sont alors les composants logiciels proxy de Web Services et Web Services pour Dispositifs type UPnP.

5.4.3 Une approche multi-*design*

Il existe plusieurs manières d'instancier des composants dans un container WComp et donc plusieurs manières de créer des applications dans WComp. Le designer graphique permet au développeur de créer les applications de manière visuelle, en réalisant des assemblages dans le container "à la main". Ces assemblages peuvent aussi être généré grâce au designer type "texte" ou à l'explorateur réseau UPnP de Intel. Quant au designer de proxy UPnP, il permet de générer des proxys pour dispositifs UPnP mais également de créer dans un container des instances de ces proxys. Il y a cinq principaux *designers* :

- un *designer* graphique d'architecture Bean4WComp,
- un *designer* de composants mixtes, i.e. de proxy de *Web Services* pour Dispositifs (*UPnPProxyWizardDesigner*),
- un *designer* type "texte",
- *Intel UPnP device spy*,
- un *designer* d'aspects d'assemblage (*AA-Designer*).

Le *designer* graphique d'architecture Bean4WComp permet par exemple de composer manuellement des assemblages de composants à partir d'une représentation graphique des flots d'événements. Il est particulièrement adapté à la description de l'application. L'application est construite en assemblant des composants sur étagères et en les reliant par diffusion d'événements.

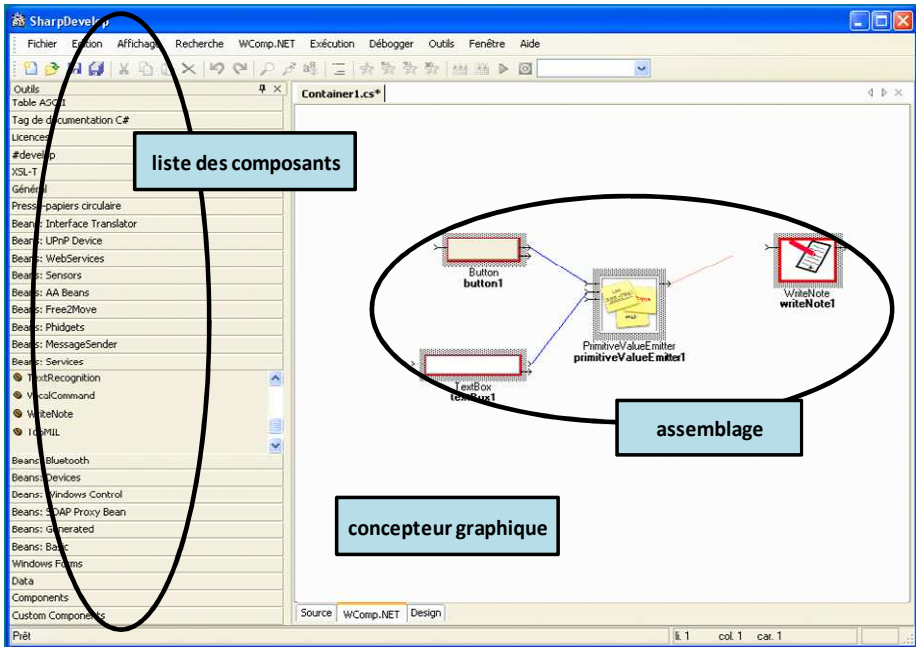


Figure 23 : **Le designer graphique d'architecture**

Pour créer une instance de composants dans un container, il faut sélectionner le type de composant dans la barre d'outil sur la gauche du concepteur (fig. 23) et cliquer ensuite dans le concepteur. Il peut être nécessaire d'adapter les propriétés du composant nouvellement créé, en utilisant le panneau de paramétrage des propriétés.

Pour créer un lien entre deux composants, il faut relier un port de sortie du premier composant à un port d'entrée du second. Lorsque les ports sont reliés, une boîte de dialogue (fig. 24) offre la possibilité de choisir l'événement qui sera émis du premier composant ; une fois l'événement sélectionné, la même fenêtre permet de choisir la méthode qui sera appelée dans le second composant (fig. 24). Enfin, si la méthode sélectionnée n'est pas compatible avec l'évènement, c'est à dire que la méthode appelée nécessite un argument de type différent de celui correspondant à l'événement source sélectionné, la fenêtre de dialogue propose alors la sélection d'une méthode pour transformer le type du paramètre (fig. 25). Il s'agit d'un rétro appel d'une méthode sur le composant émetteur.

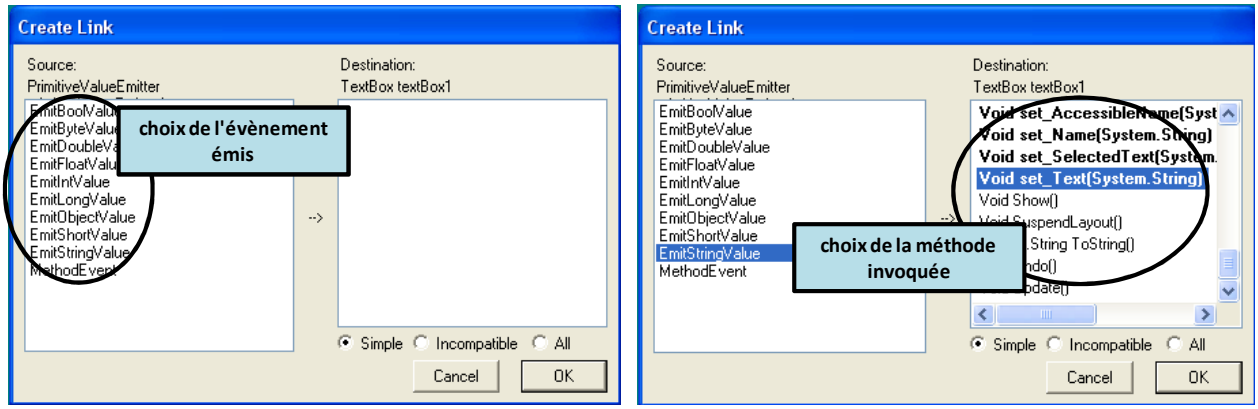


Figure 24 : **Création de lien simple**

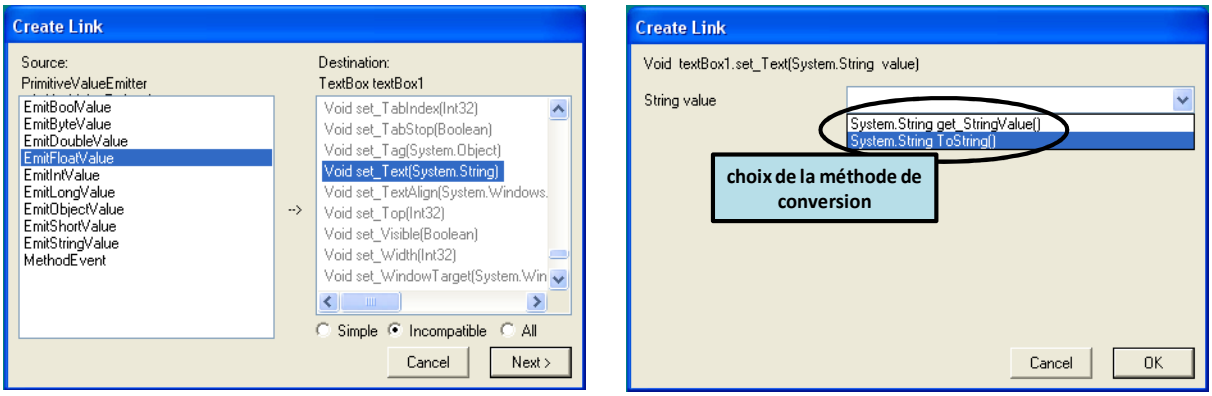


Figure 25 : **Création de lien incompatible**

Le *designer* de composants mixtes UPnP (fig. 26) permet de gérer l'apparition et la disparition des composants proxy de Web Services pour Dispositifs dans le container ; il génère et instancie automatiquement les composants proxy de dispositifs UPnP. Il gère les variations de l'infrastructure sans intervention directe de l'utilisateur. Il ne traite pas de la modification de l'assemblage de composants mais uniquement de leur instanciation ou

destruction, autrement dit, de leur ajout et de leur retrait de l'assemblage. Cette modification minimale de l'assemblage pouvant induire d'autres modifications et donc l'adaptation de l'application par composition, comme on le verra plus tard. Ce designer est configurable pour ne pas prendre en compte certains dispositifs dans l'assemblage de composants, tels que le service composite lui même. L'utilisateur ne peut agir sur le comportement de ce designer qu'en modifiant l'état réel des dispositifs physiques.

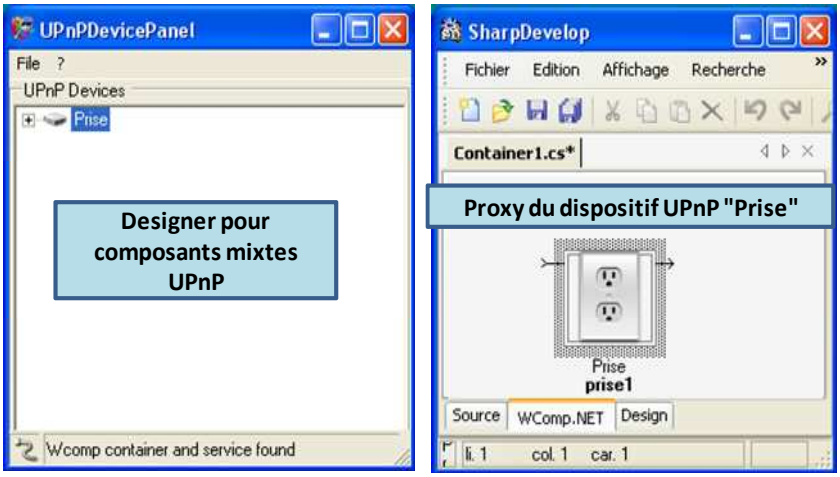


Figure 26 : ***Designer* de proxy pour *Web Service* pour dispositif**

Pour créer une instance d'un proxy pour dispositif UPnP dans un container, il est nécessaire que le container soit lui-même considéré comme un dispositif UPnP, de manière à pouvoir le manipuler au travers de son interface de contrôle. Il faut "lier" le container à un dispositif UPnP virtuel. Cette manipulation est décrite figure 27.

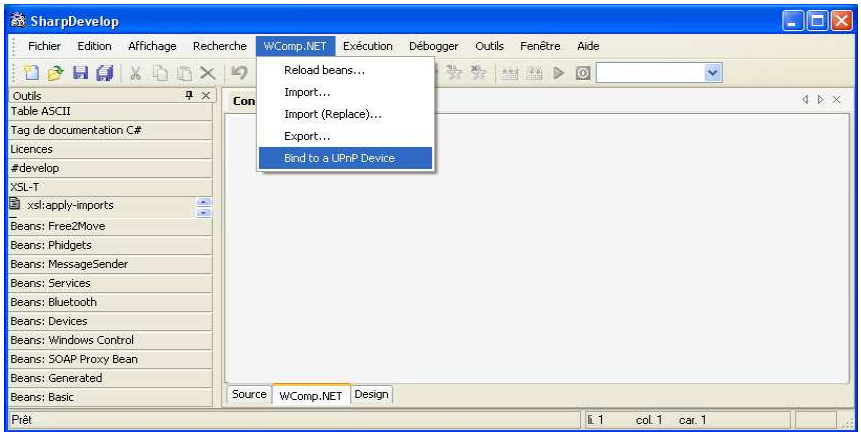


Figure 27 : **Rattachement du container à un dispositif UPnP**

Le *container* est donc vu par le *designer* comme un dispositif UPnP. Lorsqu'un dispositif UPnP apparaît sur le réseau, son proxy est créé s'il n'existe pas déjà et une instance de ce proxy apparaît automatiquement dans le *container* (voir figure 26). Ce *designer* ne permet pas de créer des liaisons entre composants.

Le designer type "texte" permet de créer ou de supprimer des instances de composants en ligne de commande. Il est également possible de créer ou de supprimer des liens entre composants.

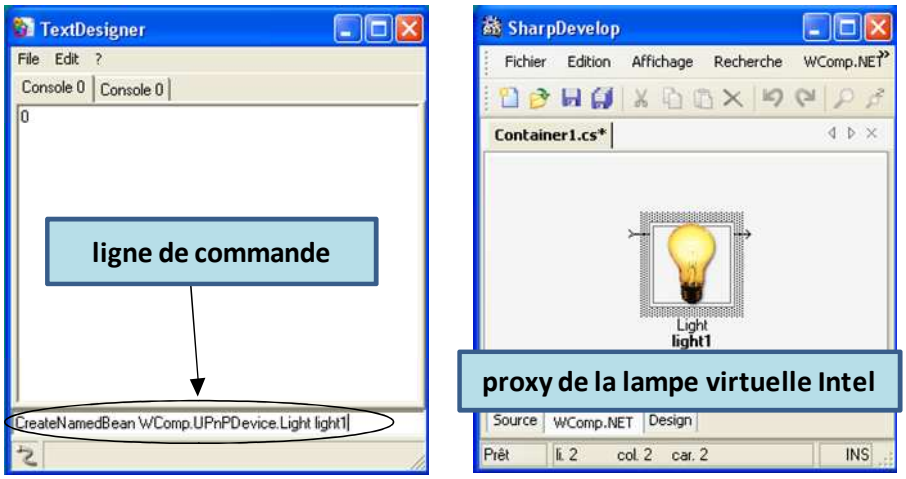


Figure 28 : ***Designer* mode texte**

Toute application dans WComp peut donc être créée ou modifiée en ligne de commande. Tout comme précédemment, le *container* doit être considéré comme un dispositif UPnP par le *designer* et la même manipulation préalable est nécessaire (fig. 27).

Un designer fourni par Intel : le *device spy*

Cet outil Intel dont l'objectif est de détecter tous les dispositifs UPnP présents sur un réseau offre également la possibilité de manipuler ces dispositifs au travers de leur interface.

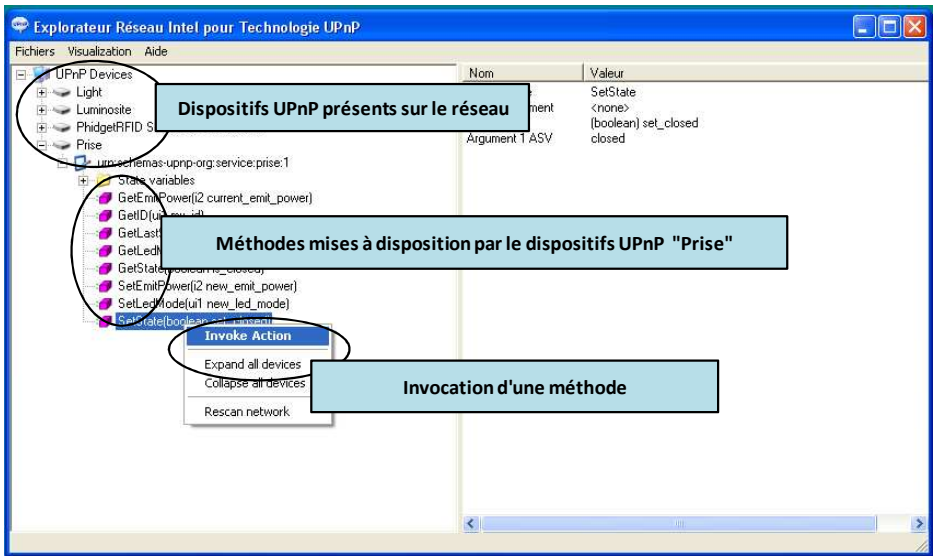


Figure 29 : **Un *Designer* : explorateur réseau UPnP de Intel**

Là encore, il faut "lier" le container à un dispositif UPnP. Cela fait, on accède facilement à l'interface de contrôle du container (fig. 29). On peut alors modifier le ou les assemblages qu'il contient en appelant les méthodes appropriées.

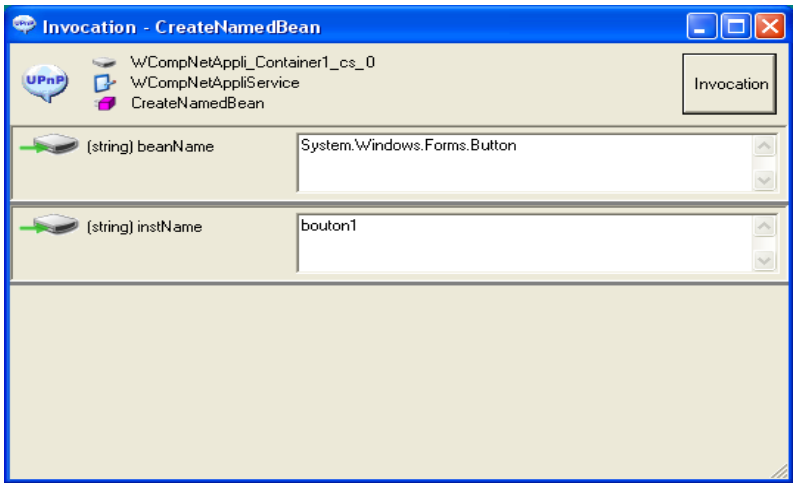


Figure 30 : **Invocation de méthode avec l'explorateur réseau UPnP Intel**

Le *AA-designer*
Le *designer* pour Aspects d'Assemblage accompagne le premier *designer* en résolvant le problème de l'adaptation des assemblages de composants en permettant de greffer des schémas d'assemblage en réponse à la variation de l'infrastructure sous-jacente [4]. Le designer pour aspects d'assemblage sera abordé plus loin.

5.4.4 *Beans* et applications

Pendant cette année de stage, j'ai été amené à développer de nombreux *beans* et à construire des applications, dont voici les principales.

Application "allumage d'une lampe"
La première application proposée est très simple. On utilise un composant de type "prise" et un composant de type "light" (voir fig. 18). Ce sont deux proxys pour dispositifs UPnP. On désire contrôler la prise (et donc la lampe qui est branchée dessus) par l'intermédiaire de la lampe virtuelle, qui sert d'interrupteur par un double clic.

.

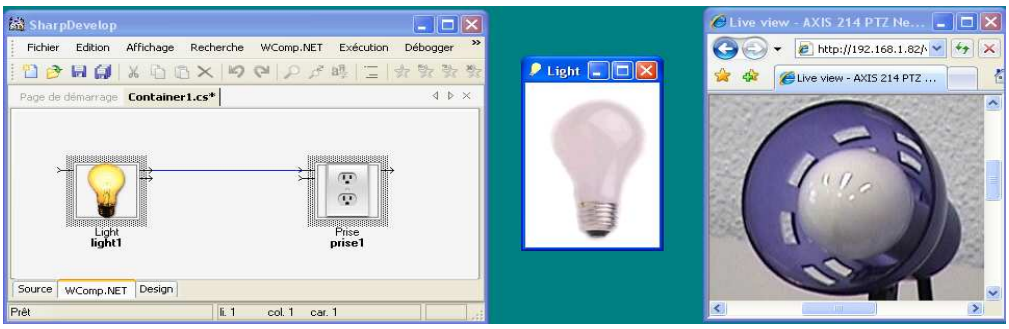


Figure 31 : **Contrôle de la prise par la lampe virtuelle : état 1**

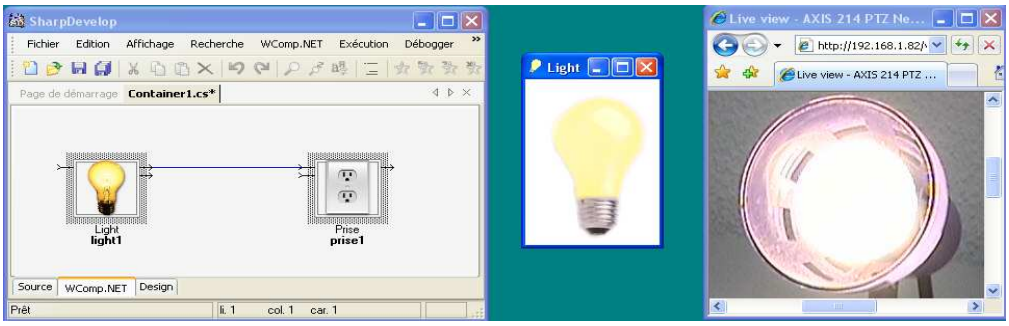


Figure 32 : **Contrôle de la prise par la lampe virtuelle : état 2**

Le nom de l'événement émis du composant "*light1*" est "*Status_Evented_NewValue*". Cet event est typé, il envoie le booléen faux ou vrai, selon l'état de la lampe virtuelle, éteinte ou allumée. La méthode appelée sur le composant "prise1" est "*SetState*". Cette méthode prend en paramètre un booléen et fait passer la prise dans l'état *true* ou *false*, c'est à dire passante ou bloquante. On aurait pu utiliser, à la place de la lampe virtuelle, un composant de type *checkBox* ou des boutons ON et OFF.

Applications utilisant des Web Services

Les deux applications suivantes font rentrer en jeu des proxys pour Web Service, le premier fournissant la météo succinte (état du ciel et température) à partir d'un nom de ville en France, le second émettant les actualités de l'université de Nice.

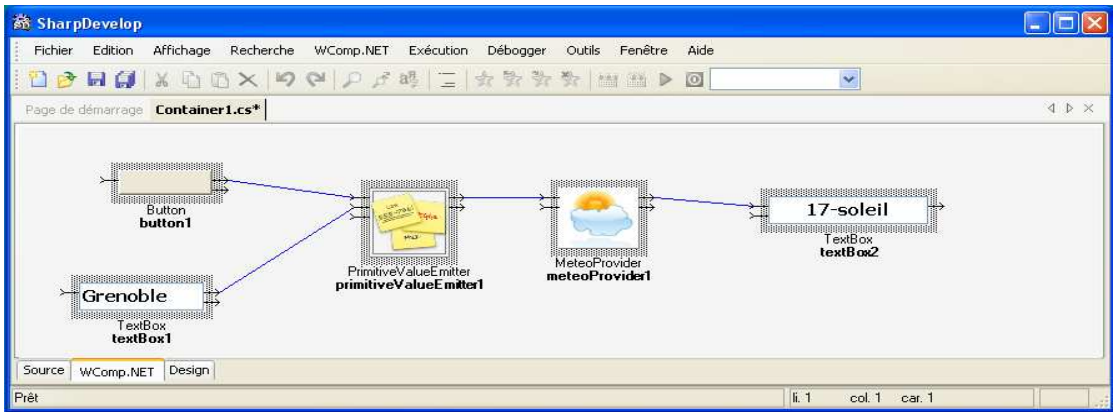


Figure 33 : **Application Web Service météo**

Il faut saisir le nom de la ville dans "textBox1" puis cliquer sur "button1". La requête est alors envoyée au WebService "meteoProvider1" et la réponse apparaît dans "textBox2"

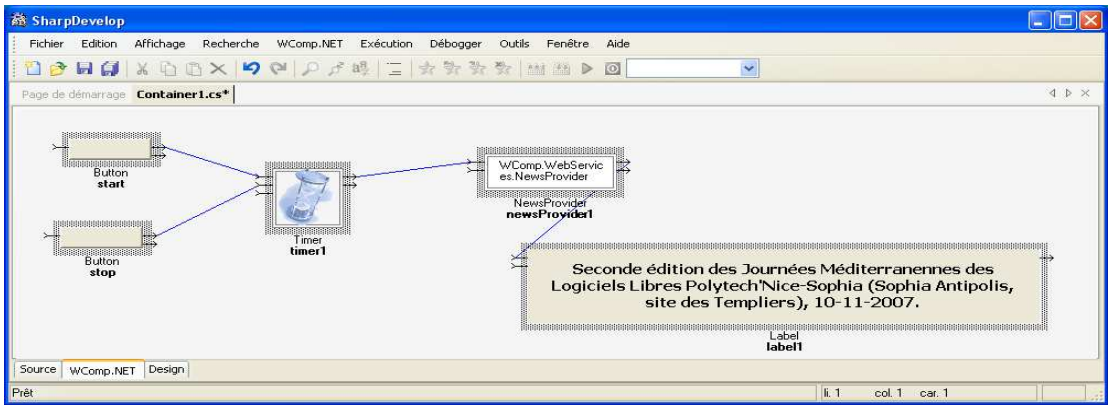


Figure 34 : **Application Web Service actualités**

Le bouton "start" déclenche le timer qui va émettre un événement toutes les x secondes. La période du timer est une propriété du composant. L'émission de l'événement provoque l'envoi d'une une requête au Web Service "NewsProvider" qui renvoit l'actualité de l'instant.

Les applications utilisant des Web Services font apparaître un problème de synchronisation. Par exemple, dans le cas d'une application utilisant deux Web Services simultanément. Si l'application est sollicitée deux fois dans un temps assez court, en tout cas plus court que le temps de réponse des deux Web Services concernés, quatre évènements seront émis en tout : deux lors de la première utilisation et deux lors de la seconde. Les résultats des requêtes risquent de ne pas être ceux escomptés, particulièrement si le temps de réponse d'un des deux Web Services est beaucoup plus court que le temps de réponse de l'autre. En effet, la communication évènementielle étant asynchrone par définition, il n'y a pas de moyen d'empêcher l'émission d'un nouvel évènement avant la réponse à la première requête. Ces problèmes de synchronisation doivent être anticipés lors de la conception des applications.

Création d'un nouveau *designer*

Cette application consiste à donner la possibilité de manipuler un container, c'est à dire de créer ou modifier des applications partir d'un autre container. Un composant a été développé pour les besoins de l'application : le bean "AssemblyHandler". Voir en annexe 5 le code de ce composant.

Il s'agit d'un *designer* de type "texte". Il permet de créer ou de supprimer un composant, ainsi que d'en modifier les valeurs des propriétés. Il permet également la création de liaisons entre composants. Comme pour la plupart des autres *designers*, il est nécessaire de "lier" le container que l'on souhaite manipuler à un dispositif UPnP virtuel. Sur la figure 35, on désire changer une propriété de la lampe virtuelle Intel : son Url. La syntaxe des commandes textuelles est décrite figure 36.

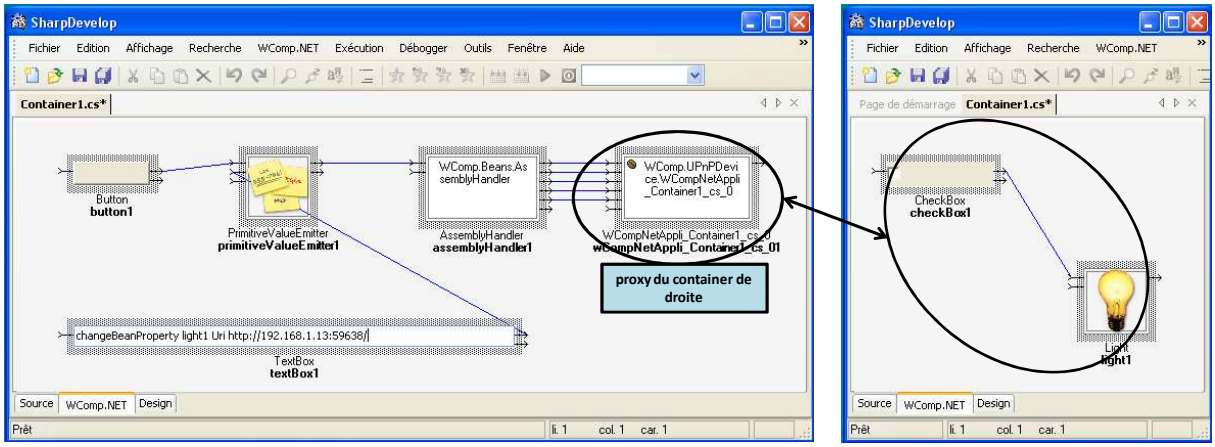


Figure 35 : ***Designer "AssemblyHandler"***

commande	argument 1	argument 2	argument 3	argument 4	argument 5
addBean	type de composant	nom du composant	NA	NA	NA
addLink	composant source	évènement source	composant cible	méthode cible	méthode de conversion d'argument (si lien incompatible)
removeBean	nom du composant	NA	NA	NA	NA
removeLink	nom de la liaison *	NA	NA	NA	NA
changeBeanProperty	nom du composant	nom de la propriété	valeur de la propriété	NA	NA
* nom d'une liaison : link-composantSource-composantCible-évènementSource-méthodeCible-(méthodeConversion-)					

Figure 36 : **Syntaxe des commandes textuelles du *designer "AssembyHandler"***

Création d'un service composite

En reprenant l'application d'allumage de la lampe vue au début de ce chapitre, on se propose de l'enrichir en faisant en sorte de pouvoir contrôler la lampe à partir d'un autre container. En d'autres termes, on veut faire un service composite de contrôle de la lampe, qui pourra être utilisé dans n'importe quel autre assemblage. Pour cela, il faut créer une méthode pour le container de l'application initiale. Un composant "*emitProbe*" est utilisé pour cet usage (figure 37). Les *beans* "BoolToString" et "StringToBool" ont été développés pour les besoins des assemblages qui vont suivre. Voir en annexe 6 le code du composant "BoolToString".

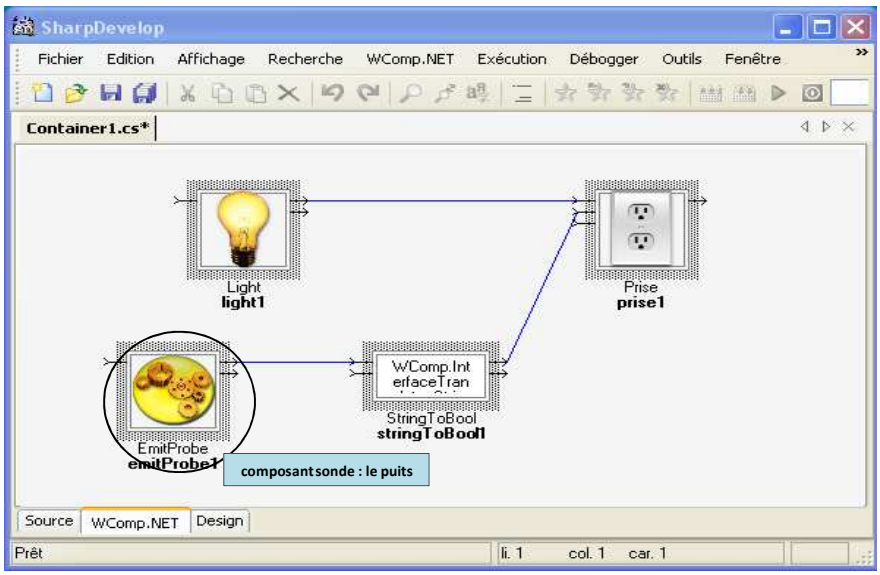


Figure 37 : **Ajoût d'une méthode sur un container (1)**

Dans un nouveau container, on crée un nouvel assemblage dans lequel on génère un proxy pour le premier container selon la procédure vue au début de ce chapitre. Une méthode est

désormais accessible via l'interface fonctionnelle du container (figure 38). La lampe peut désormais être allumée à partir du second container (figure 39).

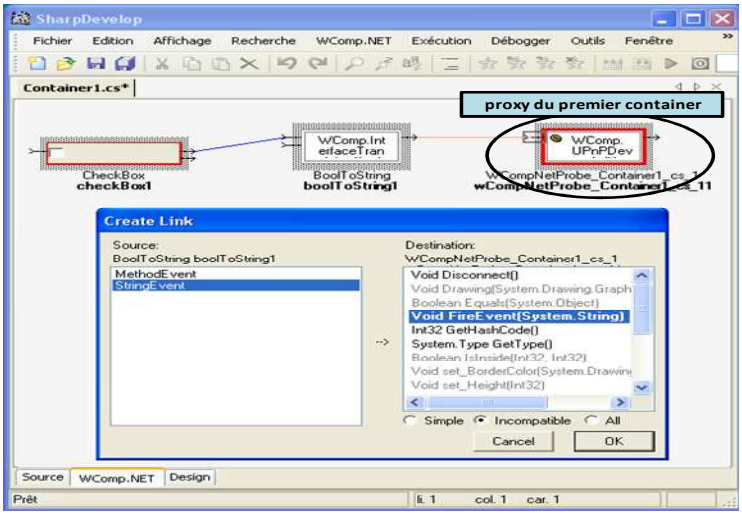


Figure 38 : **Ajoût d'une méthode sur un container (2)**

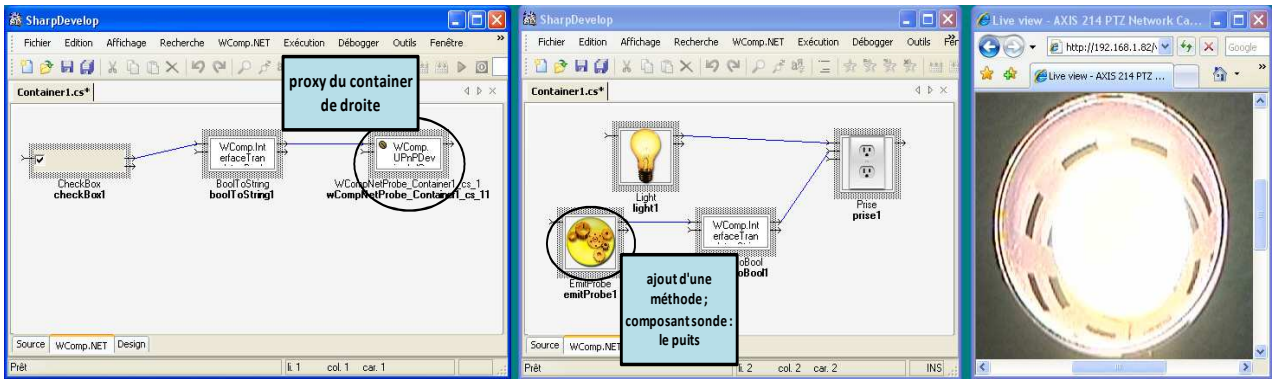


Figure 39 : **Création d'un service composite (1)**

Une autre manière de réaliser l'allumage de la lampe à partir d'un autre container est de créer un évènement qui pourra être émis à partir de ce container. Dans ce cas, le service composite créé est un service d'émission de booléen (figure 40).

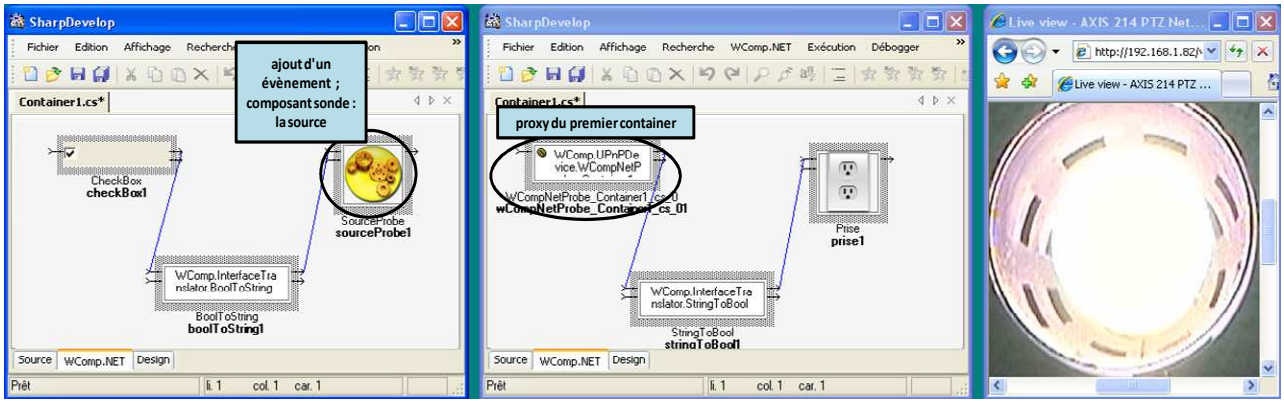


Figure 40 : **Création d'un service composite (2)**

Application "envoi de mails"
La dernière application proposée est une application permettant d'envoyer des mails et de sauvegarder des notes. Pour les besoins de cette application, trois composants ont été créés : "*stringTranslator*", "*sendMail*" et "*WriteNote*". Voir en annexe 7, 8 et 9 les codes pour ces deux composants.

Il est nécessaire de configurer le composant SendMail en indiquant l'adresse email à partir de laquelle on désire envoyer les mails, l'adresse d'un serveur SMTP accessible ainsi que le format d'envoi des mails (html ou pas). L'adresse de l'envoyeur sera mise en copie pour chaque mail envoyé, de manière à garder une trace de tous les mails. Il faut également configurer le composant WriteNote en spécifiant le répertoire dans lequel les notes seront sauvegardées, ainsi que le format de ces notes (txt ou rtf).

L'utilisateur saisit l'adresse du destinataire, les titre et corps de message, puis clique sur le bouton "SEND MAIL". Pour une note, il écrit la note dans le champ prévu et clique sur le bouton "WRITE NOTE". Sur la figure 45, on peut voir l'application telle que l'utilisateur pourra l'utiliser.

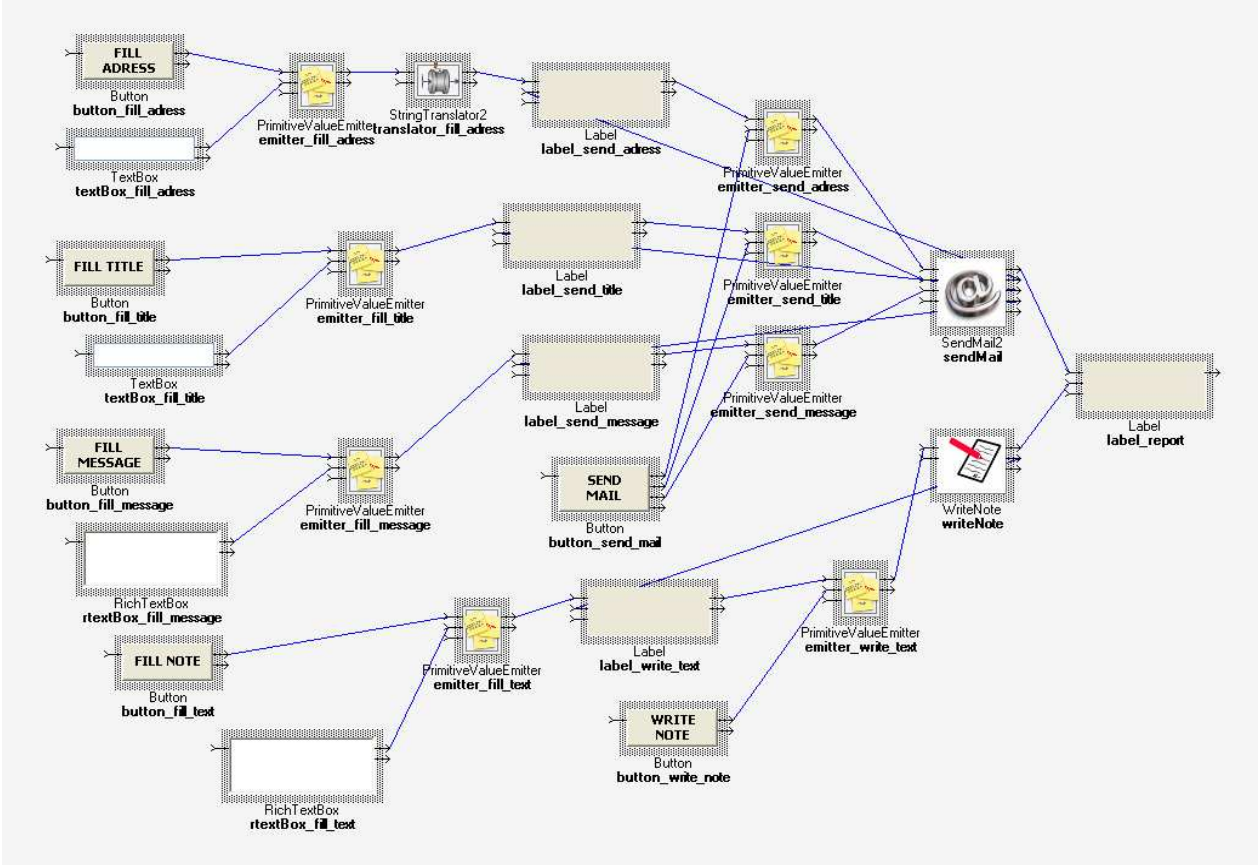


Figure 41 : **Assemblage envoi de mail et sauvegarde de notes**

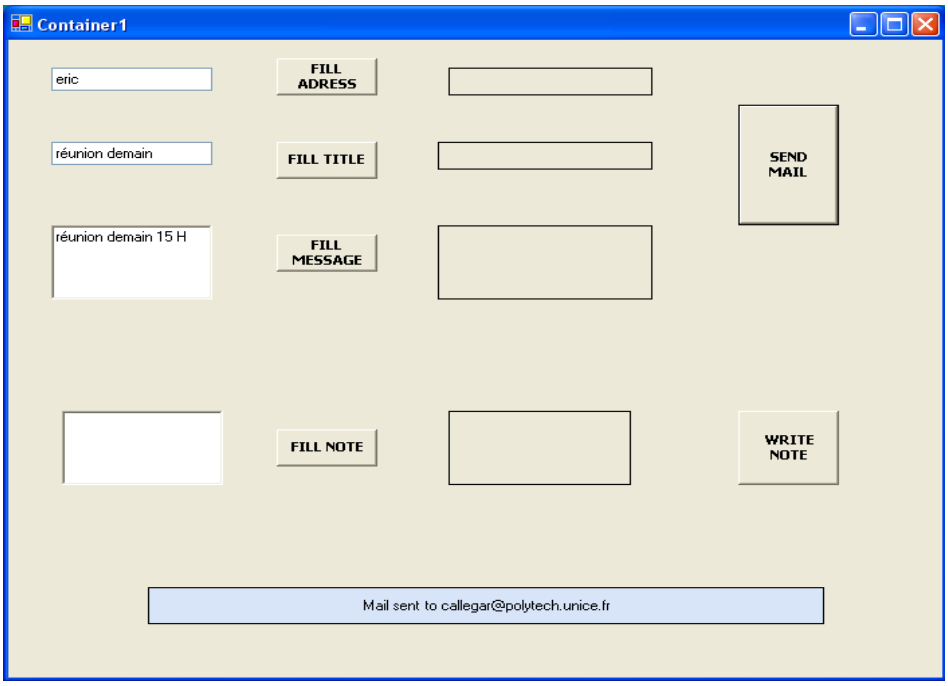


Figure 42 : **Application envoi de mail et sauvegarde de notes**

Toutes les applications citées peuvent subir des adaptations, c'est à dire des modifications des assemblages les définissant. Comme cela a été dit, on ne s'intéresse pas ici à l'aspect IHM du problème. Par exemple, dans l'application qui vient d'être présentée, l'IHM (fig. 43) restera inchangée. En revanche, l'assemblage (fig. 42) va subir des modifications pour s'adapter au contexte de l'application.

6 Adaptation des applications

Pour créer des systèmes adaptables dynamiquement à leur contexte, on utilise le paradigme de conception orientée composants. Il s'agit de concevoir les applications sous forme d'assemblages de composants. Dans ce cadre, l'adaptation d'une application va se traduire par une modification dynamique de l'assemblage et par l'introduction de composants de contrôle. Pour exprimer ces adaptations, on introduit la notion **d'aspect d'assemblage**.

6.1 La programmation orientée aspects (P.A.O.)

Les techniques de conception logicielles actuelles tentent de rendre les applications plus modulaires, les modules étant censés être indépendants les uns des autres car gérant des aspects différents du système conçu. C'est le principe même de la programmation orientée objet où le logiciel est découpé en unité : les objets. Dans la pratique cependant, on s'aperçoit que ces couches logicielles sont en fait intimement liées : c'est l'entrecroisement des aspects techniques. Ainsi, une couche logicielle initialement dédiée à gérer la logique métier applicative se retrouve dépendante d'autres modules gérant des aspects techniques.

Il a donc été introduit un nouveau paradigme permettant de gérer les aspects transversaux d'une application : la programmation orientée aspect (P.A.O. ou A.O.P. pour Aspect Oriented Programming). L'inversion de contrôle mise en œuvre par la programmation par aspect permet d'extraire les dépendances entre objets ou modules concernant des aspects techniques entrecroisés. Ces dépendances sont gérées depuis l'extérieur de ces modules. Elles sont spécifiées dans des composants du logiciel nommés aspects.

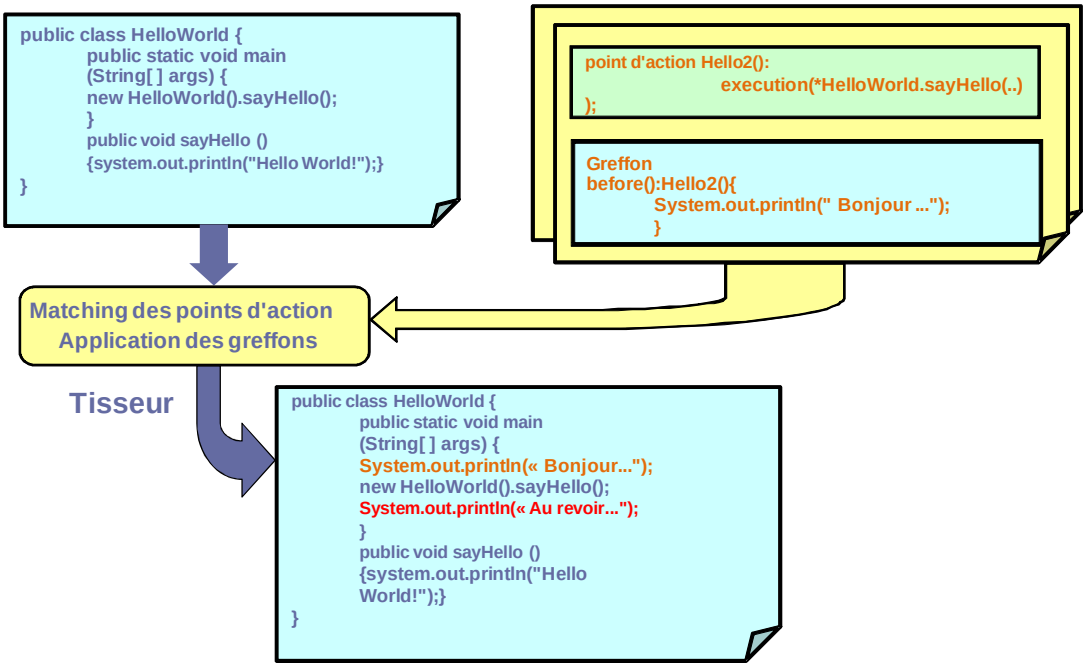


Figure 43 : **Programmation orientée aspect : mécanisme de tissage**

Un aspect permet de spécifier :

- Des points d'action ou points de coupe (*pointcut* en anglais), qui définissent les points de jonction satisfaisants aux conditions d'activation de l'aspect, donc le ou les moments où l'interaction va avoir lieu.
- Des greffons, c'est-à-dire les programmes (*advice* en anglais) qui seront activés avant, autour de ou après les points d'action définis.

Des exemples de points de jonction, de greffons et de tissage sont décrits figure 43. La programmation orientée aspects permet donc, grâce à un mécanisme de tissage et aux définitions de points d'action et de greffons, d'insérer du code dans un programme existant. C'est par analogie avec les aspects de la P.A.O. qu'est né le paradigme d'aspect d'assemblage.

6.2 Les aspects d'assemblages

6.2.1 Fonctionnement et définitions

Un aspect d'assemblage (aussi appelé AA) est constitué, comme un aspect tel que défini précédemment, de deux parties : un point de coupe et un greffon. Cependant l'application d'un aspect d'assemblage conduit à la modification d'un assemblage de composants, et non à l'ajout de code, comme dans la P.A.O.

La première différence avec la P.A.O. est bien sur le point de coupe. Les points de coupe sont des règles et non des commandes dans un langage de programmation. Ces règles permettent d'établir la liste des points de jonction dans l'assemblage initial, où les greffons s'appliqueront. Le greffon quant à lui ne consiste plus en une ou plusieurs lignes de code, mais en un sous-assemblage qui viendra se "connecter" à l'assemblage initial, aux points de jonction, comme il est décrit figure 44.

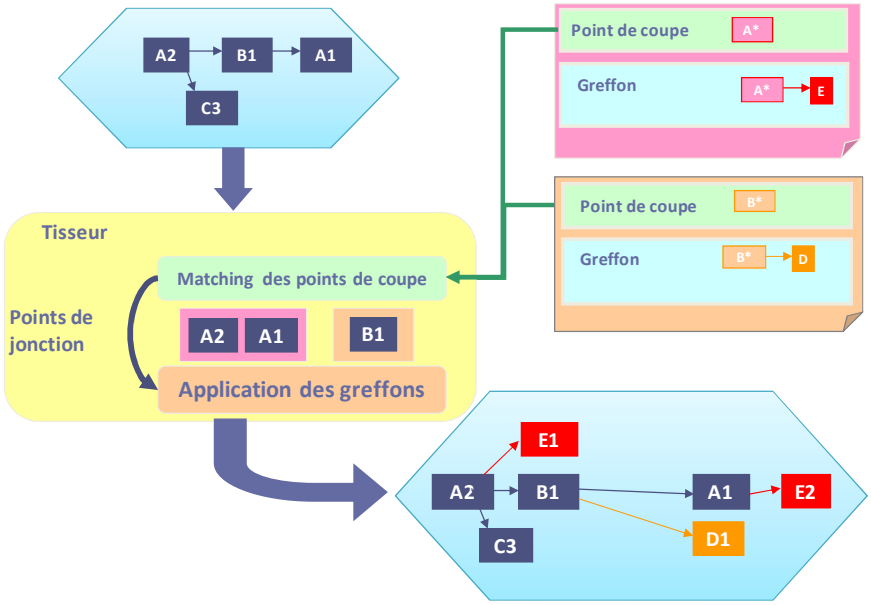


Figure 44 : **Aspect d'assemblage : mécanisme de tissage**

Sur l'exemple ci-dessus, les règles définissant les points de jonction sont A* et B*, les points de jonction sont A1 et A2 pour le premier point de coupe et B1 pour le second, et enfin les greffons sont E1, E2 et D1.

Le concept d'aspect d'assemblage se définit donc en trois volets :

- le point de coupe ou l'expression des points de jonction dans l'assemblage où l'aspect d'assemblage est introduit,
- le greffon ou la représentation de l'assemblage qu'on introduit (il s'agit ici d'un schéma d'assemblage),
- le tisseur ou la méthode pour composer les assemblages à introduire dans l'assemblage existant.

Le premier volet identifie les points d'ancrage de la modification d'assemblage produite par le schéma d'assemblage de l'aspect. Le deuxième volet décrit l'assemblage à introduire, en utilisant des opérateurs bien spécifiques au schéma d'assemblage de l'aspect. Pour un domaine particulier, il n'est pas utile de spécifier le troisième volet qui est fixé une fois pour toute. Ce volet consiste à traduire les spécifications décrites dans le second volet en un ensemble de modifications élémentaires qui vont entrainer une modification de l'assemblage d'origine. [12]

6.2.2 Points de coupe

Les points de coupe de l'aspect décrivent par une expression régulière les points de jonction où le schéma d'assemblage doit se positionner. Ces expressions régulières portent sur les identifiants des instances de type de composant et sur leurs ports. Un point de coupe est donc ici représenté par une expression régulière. Un point de jonction est un des identifiants de l'assemblage désigné par l'expression régulière.

(<composant>|<événement>|<méthode>) = <expression régulière>

6.2.3 Greffons

Les greffons des Aspects d'Assemblage reposent sur le schéma d'assemblage correspondant. Dans la version actuelle de WComp, on s'appuie sur une description des schémas d'assemblage basée sur le langage de spécification d'interactions ISL. ISL (Interaction Specification Language) est un langage de description de schémas d'interactions entre des composants. ISL4WComp étend ISL pour la prise en compte des interactions basées sur des événements. Ces langages ont la propriété d'être composables c'est-à-dire que plusieurs schémas peuvent se composer et fusionner en un seul schéma combinant leur comportement. Ces propriétés sont utilisées dans les aspects d'assemblage, notamment la commutativité, c'est à dire que l'ordre d'application des aspects n'influence pas le résultat de leur composition.

Au sein d'ISL4WComp, deux façons de modifier le comportement d'une application sont définies :

- d'une part, sur un appel de méthode,
- d'autre part, sur la diffusion d'un événement.

Dans tous les cas, on y associe un nouveau comportement programmé avec les opérateurs du langage (voir fig. 45).

Opérateurs	Description
;	exprime une séquence
||	exprime un ordre indifférent entre les composants
if … else …	exprime une condition
call	désigne l'évènement ou l'appel du point de jonction
delegate	désigne l'émission d'un évènement ou un appel
^	exprime l'émission d'un évènement

Figure 45 : **Principaux opérateurs du langage ISL4WComp**

La structure d'un aspect d'assemblage est la suivante :

<point de jonction> : <composant>.<événement>|<composant>.<méthode>

schema <nom> (<composants>, ...) {
<point de jonction> {<comportement programmé ISL4WComp>}
}

Un aspect d'assemblage repose donc sur des ensembles de règles. Une règle porte sur des points de jonction qui sont soit l'émission d'un événement, soit un appel de méthode. Certains mots-clés (voir fig. 45) du langage, comme *call* et *delegate*, permettent de contrôler la manière dont les schémas d'assemblage des aspects d'assemblage vont se composer. La définition d'aspect d'assemblage est donc un moyen de représenter des modifications d'assemblages de composants en y apportant la possibilité de les fusionner automatiquement. Des exemples d'écriture et de composition d'aspects d'assemblage seront décrits par la suite.

6.2.4 Tisseurs

La technique d'application des aspects d'assemblage utilisée est constituée d'un mécanisme de tissage d'aspects dans le domaine de la programmation logicielle orientée aspect A.O.P. appliquée aux composants logiciels pour modifier l'assemblage qui implémente l'application.

Dans le cadre du tissage d'aspect d'assemblage, deux cas de figure se présentent, selon le greffon. Le premier cas – le plus simple – se produit lorsque les deux greffons n'ont pas de point de jonction en commun. Les deux sous-assemblages s'insèrent alors indépendamment l'un de l'autre dans l'assemblage de départ. Le second cas se produit lorsque les greffons ont au moins un point de jonction en commun. Il y a alors fusion des deux aspects d'assemblage. Il s'agit en fait de fusionner les algorithmes des deux greffons pour en déduire un nouvel algorithme qui correspondra au nouveau greffon à insérer dans l'assemblage initial. La fusion des schémas d'assemblage correspond à la mise en œuvre des travaux formels sur la composition des règles pour ISL4WComp et des règles de logique de fusion, ainsi qu'un algorithme de fusion ont été définis. Quelques exemples de ces règles sont données figure 46.

Enfin, le mécanisme de fusion des aspects d'assemblage respecte trois propriétés fondamentales que sont la commutativité, l'associativité et l'idempotence (fig. 47). Le respect de ces propriétés garantit en pratique que le résultat de l'adaptation ne dépend pas de l'ordre d'application des aspects d'assemblage.

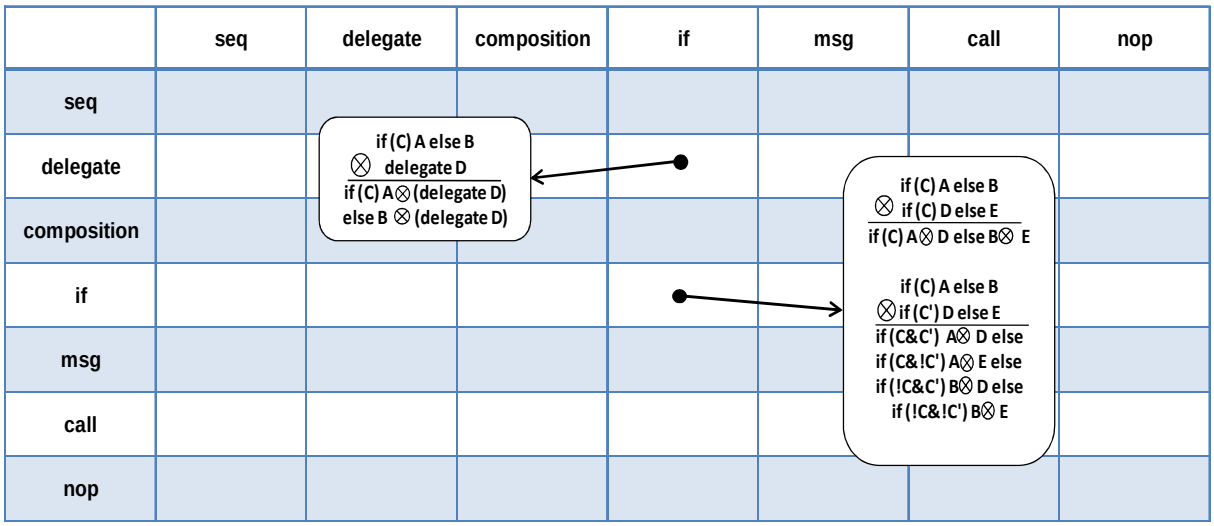


Figure 46 : **Tableau de logique de fusion des aspects d'assemblage**

Commutativité : AA0 ⊗ AA1 = AA0 ⊗ AA1
Associativité : (AA0 ⊗ AA1) ⊗AA2 = AA0 ⊗ (AA1 ⊗ AA2)
Idempotence : AA0 ⊗ AA0 = AA0

Figure 47 : **Propriétés fondamentales de la fusion des AA**

6.2.5 Le *designer* d'aspects d'assemblages

Le *designer* étudié ici est le *designer* ISL4WComp, qui sera appelé par la suite "AA *designer*". Son IHM est décrite figure 48. Elle se compose de trois parties. La première (fig. 48, A) contient la liste de tous les aspects d'assemblage disponibles. La colonne "Act" permet d'activer un aspect d'assemblage. Plusieurs d'entre eux peuvent être activés simultanément. La colonne "Sel" indique quels aspects d'assemblage sont actuellement appliqués ou selectionnés. Il est intéressant de noter qu'un aspect peut être activé sans pour autant être appliqué. Enfin, la colonne "Cur" indique simplement quel aspect est visible dans les colonnes B et C. les deuxième et troisième parties de l'IHM, respectivement les parties B et C de la figure 48 décrivent quant à elles un aspect d'assemblage. La partie B décrit les points de coupe de l'aspect d'assemblage, c'est à dire les points de jonction ou d'application des schémas d'assemblages qui sont décrits dans la partie C.

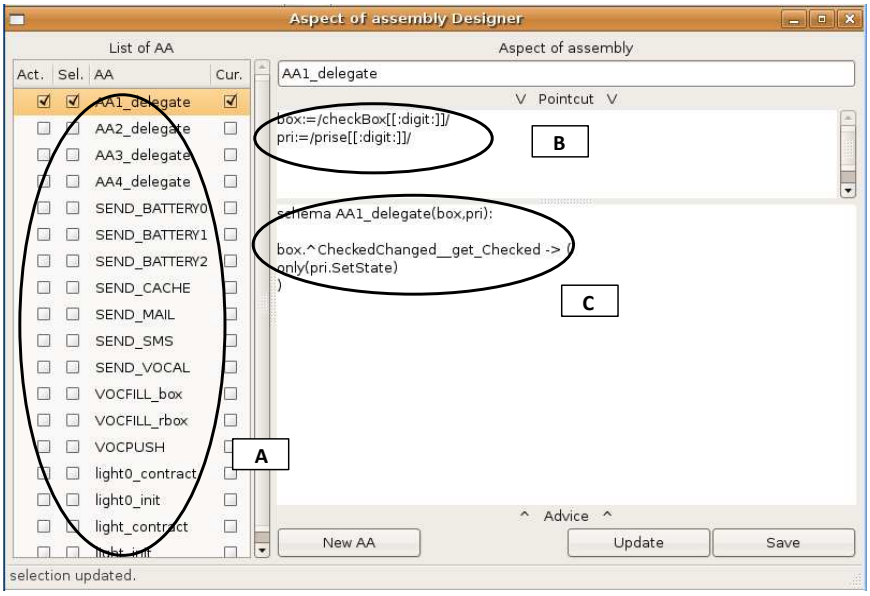


Figure 48 : **IHM du AA *designer***

L'application de l'aspect d'assemblage décrit figure 48 nécessite la présence d'un composant dont le nom est composé de la chaîne de caractère "*checkBox*" suivie d'un caractère numérique, et d'un composant dont le nom est composé de la chaîne de caractère "prise" suivie d'un caractère numérique. Sur la figure 49 sont montrés plusieurs assemblages pour lesquels l'aspect d'assemblage s'applique ou pas.

Dans le deuxième cas (fig. 49, 2), l'aspect ne s'applique pas parce qu'il y a incertitude sur le composant "checkBox" et le designer ne peut pas déterminer où l'aspect doit s'appliquer. Dans les premier et troisième cas (fig. 49, 1 et 3), l'aspect s'applique sans problème.

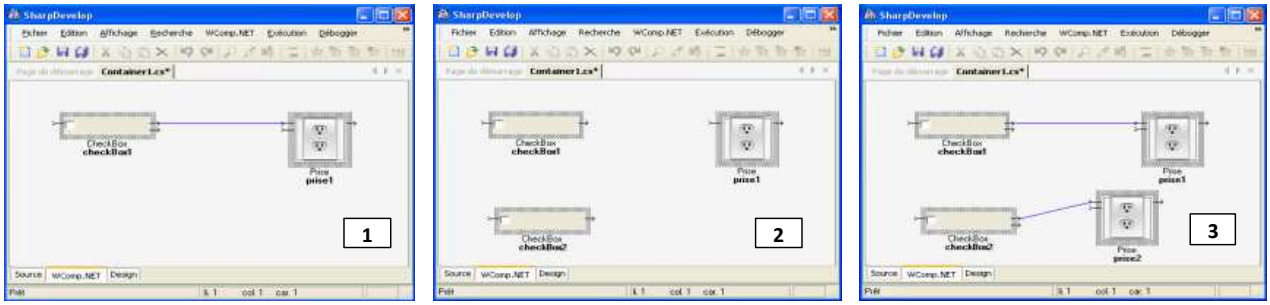


Figure 49 : **Exemples d'application d'un aspect d'assemblage**

Le AA designer est capable de générer automatiquement des composants tels que les composants "par" ou "if", par exemple, lorsque la logique imposée par le schéma d'assemblage l'exige, comme on le verra bientôt. Ce designer mérite bien son appellation puisqu'il permet de créer des liens, ainsi que des composants. On peut donc théoriquement créer des applications avec ce *designer*. Néanmoins, ce n'est pas sa fonction. Il a été conçu pour permettre les adaptations d'applications existantes à des changements de contextes.

6.3 Différents types d'adaptation

Un changement de contexte pour une application peut se traduire de différentes manières. Il peut s'agir de l'utilisateur de cette application qui évolue dans son environnement, ou encore de l'environnement lui-même qui subit un changement. Enfin, on admettra dans les changements de contexte, toute contrainte nouvelle que l'on veut imposer à l'application. Ces contraintes peuvent par exemple traduire des contrats de qualité de service.

6.3.1 Exemple de contrats de qualité de service

Exemple 1 : contrat de type "*time out*" sur un *Web Service*

On désire imposer un contrat à l'application décrite figure 33, c'est à dire l'application mettant en jeu le Web Service meteo. Il s'agit de faire remonter une erreur lorsque la réponse à la requête envoyée n'arrive pas. Pour cela, on impose un "*timeout*" à l'application. Le contrat imposé se traduit par l'application de l'aspect d'assemblage décrit figure 50. L'assemblage modifié par l'application de l'aspect est décrit figure 51.

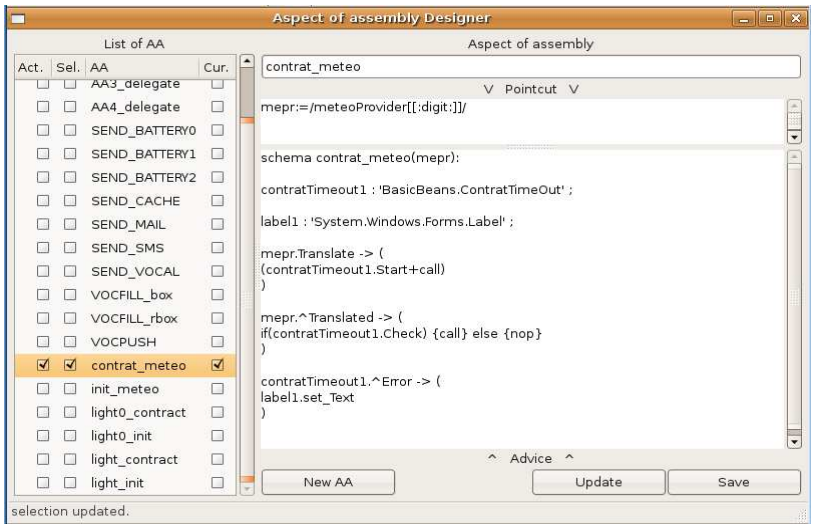


Figure 50 : **Aspect d'assemblage contrat "timeout" sur Web Service**

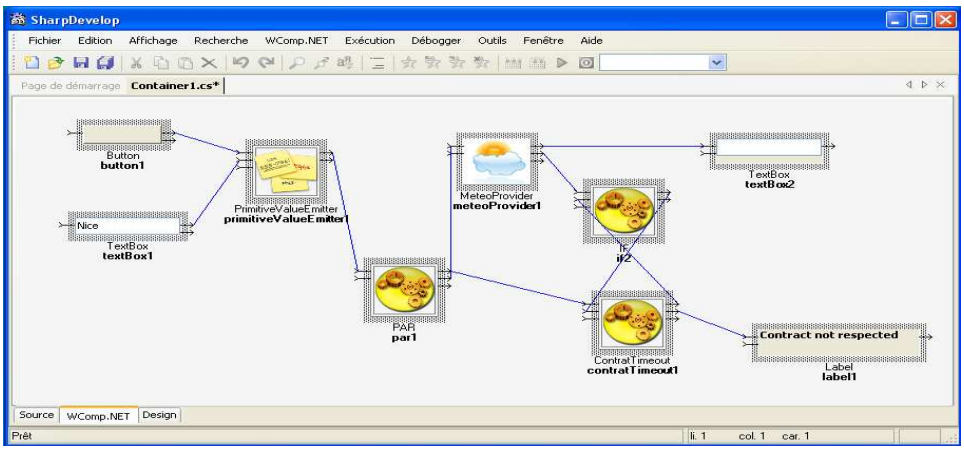


Figure 51 : **Contrat "timeout" sur un Web Service**

Le AA designer a subi, pendant le déroulement du stage, une évolution importante. En effet, le constat a vite été fait que l'adaptation des applications nécessitait l'ajout de nouveaux composants dans l'assemblage et le AA designer devait être capable de les instancier. Ces composants sont indiqués dans le schéma de l'aspect d'assemblage (partie *advice* ou greffon). Ces composants sont appelés composants locaux et ils sont détruits lorsque l'aspect d'assemblage n'est plus sélectionné.

Cette nouvelle fonctionnalité du AA designer est illustrée sur l'exemple 1. Plusieurs composants ont été instanciés lors de l'application de l'aspect d'assemblage figure 50 D'abord un composant de type "ContratTimeout" et un autre de type "Label". Ces composants sont nécessaires au contrat que l'on veut imposer à l'application. Si la réponse à la requête au Web Service dépasse la période indiquée dans les propriétés du composant *contratTimeout1*, ce dernier envoie un message d'erreur par l'intermédiaire du composant label1. D'autres composants sont instanciés automatiquement par le AA designer pour gérer les assertions conditionnelles (if … then … else) et le parallèlisme ou la séquentialité des évènements (composants "par" ou "seq").

Une première difficulté apparaît néanmoins. En effet, l'application n'est pas directement utilisable une fois l'aspect d'assemblage appliqué. Il est nécessaire de configurer le composant *contratTimeout1*, c'est à dire de modifier la valeur de sa propriété "*Timeout*" avec la valeur désirée.

Exemple 2 : contrat sur l'application de contrôle de la prise (voir fig. 34 et 35)

On impose sur l'allumage de la lampe une contrainte de luminosité. La prise sera "passante" uniquement si le niveau de luminosité est suffisamment faible. Le contrat imposé se traduit par l'application de l'aspect d'assemblage décrit figure 52. L'assemblage modifié par l'application de l'aspect est décrit figure 53. Pour s'appliquer, l'aspect d'assemblage nécessite, en plus des composants *light1* et prise1 présents initialement, la présence d'un capteur de luminosité (voir fig. 19). Les composants locaux instanciés lors de l'application de l'aspect sont un composant de type seuil ou *Threshold*, un composant de type *PrimitiveValueEmitter* et deux composants Windows de type *TextBox*.

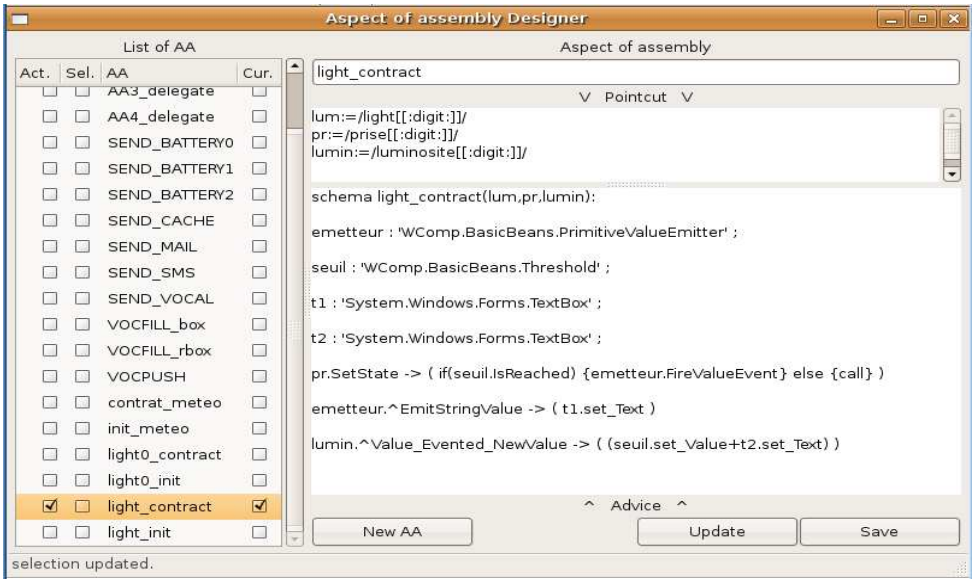


Figure 52 : **Aspect d'assemblage : contrat luminosité sur la prise**

Il faut remarquer ici que l'aspect d'assemblage peut être présélectionné. Dès l'apparition du capteur de luminosité sur le réseau, l'aspect d'assemblage s'appliquera. Si l'aspect est sélectionné tandis que le capteur est déjà présent dans l'assemblage (en plus des composants "lightX" et "priseX"), il s'appliquera bien sur immédiatement.

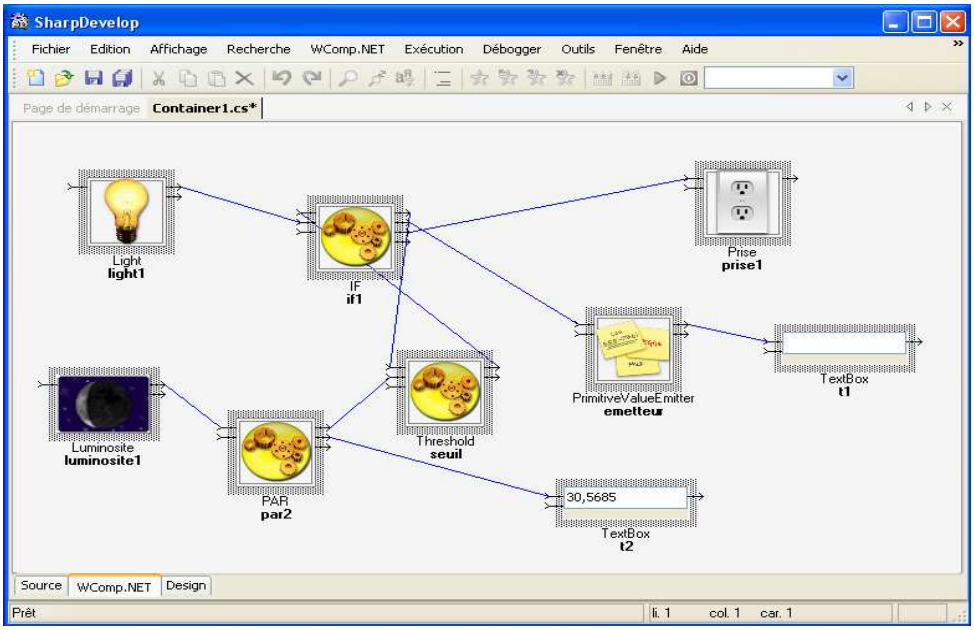


Figure 53 : **Contrat luminosité sur la prise**

De même que dans l'exemple précédent, il faut configurer le composant *threshold* avec une valeur de seuil, ainsi que le composant primitiveValueEmitter avec un message indiquant le non respect du contrat.

A la vue des deux précédents exemples, on peut affirmer que la possibilité d'accéder en écriture aux propriétés d'un composant est une évolution nécessaire du *designer* d'aspect d'assemblage.

6.3.2 L'environnement changeant

L'exemple étudié ici est une application d'envoi de messages multi dispositifs (voir fig. 54). Cette application illustre bien l'adaptation des applications à un environnement fluctuant. Elle doit s'adapter à son contexte, de manière à assurer pour l'utilisateur la continuité du service "messagerie". Pour ce faire, différents modes de fonctionnement sont définis. Les messages peuvent être envoyés soit par mail, soit par SMS. Il est également possible de stocker les messages dans une mémoire tampon (cache) pour être envoyés plus tard.

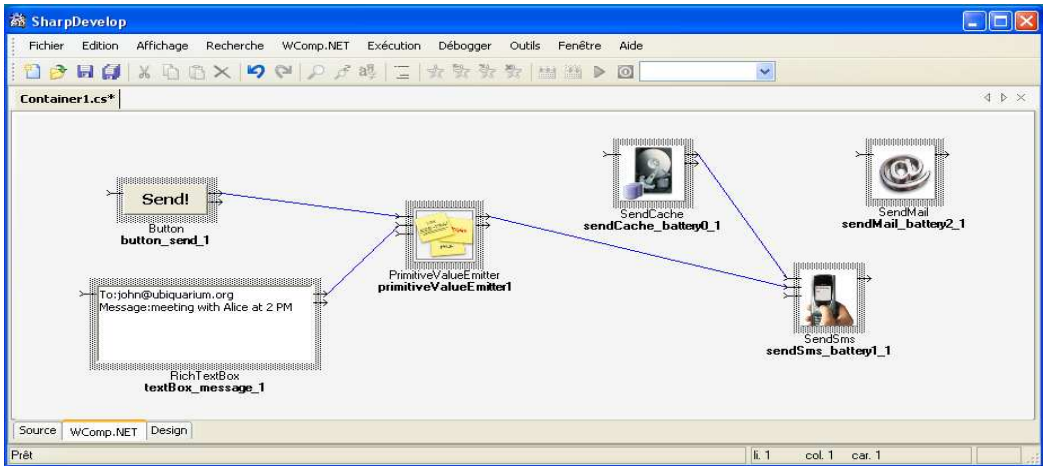


Figure 54 : **Application d'envoi de messages multi dispositifs**

Plusieurs critères vont déterminer le choix de l'un ou l'autre des modes d'envoi du message. Le premier critère est évidemment la présence ou non des dispositifs d'envoi sur le réseau. Le second critère choisi est le niveau de batterie du dispositif hébergeant l'application (ordinateur portable, PDA, téléphone portable). Chacun de ces critères correspond à une famille d'aspects d'assemblage.

La première famille d'aspects d'assemblage concerne la présence des dispositifs sur le réseau. Elle est composée des aspects d'assemblage nommés "SEND_XXX" avec XXX appartenant à l'ensemble {CACHE, MAIL, SMS} (voir fig. 55). A tout moment, un au moins de ces trois aspects doit être activé. Si les deux modes d'envoi sont disponibles, un et un seul doit être choisi. S'ils sont tous les deux absents, l'aspect correspondant au stockage des messages en mémoire cache sera activé.

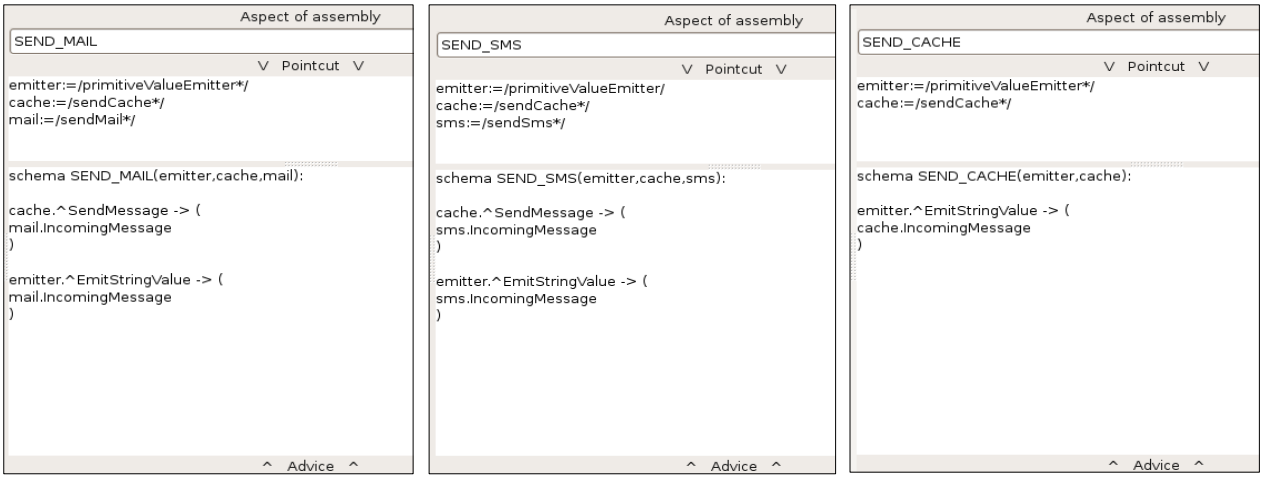


Figure 55 : **Aspects d'assemblage, famille A**

Les aspects d'assemblage nommés "SEND_BATTERYx" (x = 0,1,2) constituent la famille d'aspects d'assemblage correspondant au second critère (voir fig. 56). Ils permettent de choisir le mode d'envoi de messages en fonction du niveau de batterie, certains modes étant plus économiques que d'autres en termes de consommation d'énergie. Si le niveau de batterie est assez haut, aucun des aspects n'est activé. A partir d'un niveau suffisamment bas, un des trois aspects doit être activé, selon le niveau de batterie courant. "SEND_BATTERY0" est le mode pour le niveau de batterie le plus bas et correspond au stockage des messages en mémoire tampon.

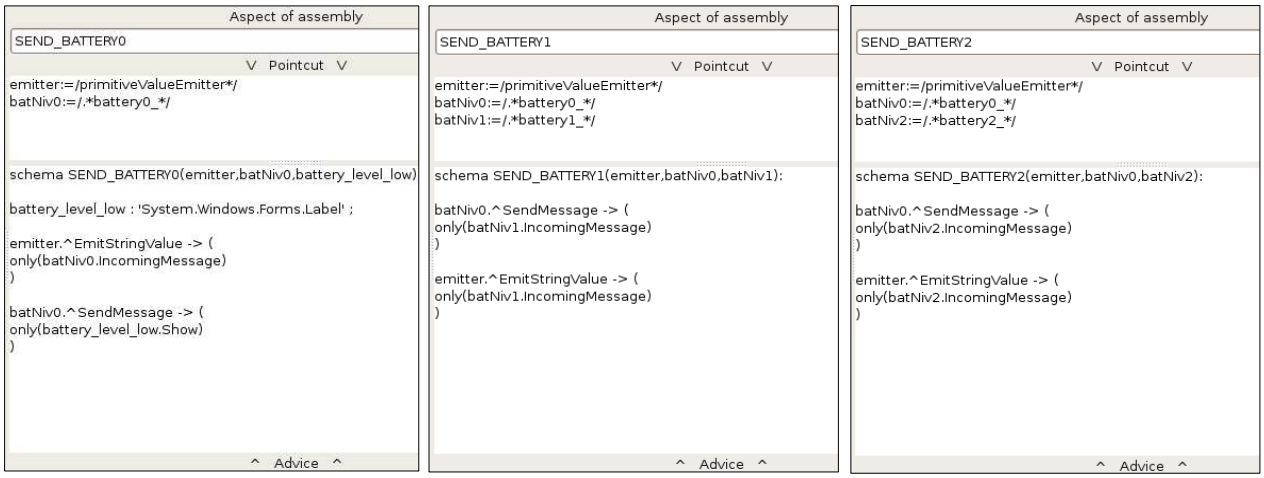


Figure 56 : **Aspects d'assemblage, famille B**

Il faut par conséquent gérer la fusion de deux aspects d'assemblage, l'un étant activé en fonction du premier critère, et le second étant activé en fonction du second. Deux scénarii de fusion vont maintenant être examinés.

Scénario 1 : le niveau de batterie est en dessous d'un certain seuil et l'aspect "SEND_BATTERY1" (famille B) est activé. Ce niveau de batterie correspond à l'envoi des messages par SMS. Pour la famille A, tous les dispositifs d'envoi sont disponibles et par choix, l'aspect "SEND_MAIL" est activé.

La fusion de ces deux aspects donne l'assemblage figure 54. Au final, le dispositif d'envoi de messages par SMS est sélectionné et ce, malgré l'activation de l'aspect "SEND_MAIL". En examinant le code des schémas des aspects, on constate que la commande "*only*" est utilisée. Cette commande permet de gérer les conflits d'émission d'évènements à partir d'un même composant. En effet, si deux aspects ont pour règles respectives l'émission d'un même évènement provoquant respectivement l'appel de deux méthodes sur deux composants différents, la commande "*only*" permet d'établir une priorité sur l'émission d'évènement. Ainsi, lors de la fusion des deux aspects, le schéma contenant la commande "*only*" sera pris en compte et l'autre sera ignoré (voir fig. 57).

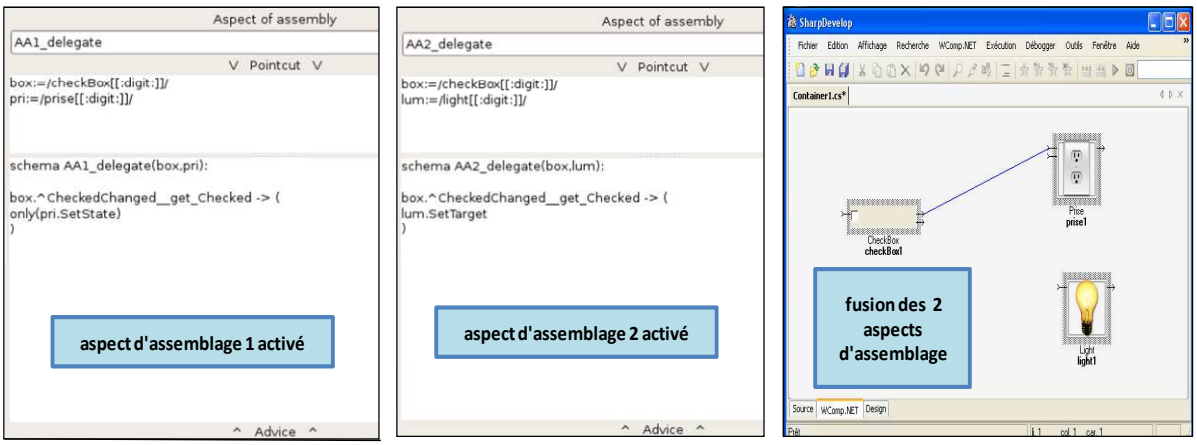


Figure 57 : **Fusion des aspects d'assemblage : commande "*only*"**

Scénario 2 : le niveau de batterie est en dessous d'un certain seuil et l'aspect "SEND_BATTERY2" (famille B) est activé. Ce niveau de batterie correspond à l'envoi des messages par mail. Pour la famille A, le dispositif d'envoi par mail n'est pas présent sur le réseau et l'envoi par SMS est possible. Par conséquent, l'aspect "SEND_SMS" est activé.

Ce cas est plus simple que le précédent. Il n'y aura pas de fusion à proprement parler. En effet, comme le service d'envoi par mail n'est pas disponible, l'aspect d'assemblage "SEND_BATTERY2", bien qu'activé, ne sera pas sélectionné. L'aspect d'assemblage "SEND_SMS" est donc sélectionné.

Pour que la continuité de service soit assurée, il faut vérifier que toutes les possibilités de fusion débouchent sur une application qui fonctionne. Aucun message ne doit être perdu et un seul mode d'envoi doit être "actif" à un instant donné. Toutes les combinaisons possibles et leurs résultats sont inventoriées figure 57.

		Aspect d'assemblage activé : famille A			
		SEND_MAIL (SMS présent)	**SEND_MAIL (SMS absent)**	**SEND_SMS**	**SEND_CACHE**
aspect d'assemblage activé : famille B	**SEND_BATTERY0**	pas d'envoi (messages en mémoire tampon)	pas d'envoi (messages en mémoire tampon)	pas d'envoi (messages en mémoire tampon)	pas d'envoi (messages en mémoire tampon)
	SEND_BATTERY1	envoi par SMS	envoi par mail	envoi par SMS	pas d'envoi (messages en mémoire tampon)
	SEND_BATTERY2	envoi par mail	envoi par mail	envoi par SMS	pas d'envoi (messages en mémoire tampon)
	AUCUN AA ACTIVE	envoi par mail	envoi par mail	envoi par SMS	pas d'envoi (messages en mémoire tampon)

Figure 58 : **Fusion des aspects d'assemblage : tableau récapitulatif**

L'adaptation dynamique d'une application reste contrôlable lorsque deux critères d'adaptation sont en jeu. En effet, en construisant habilement les aspects d'assemblage, on arrive facilement à obtenir le résultat voulu lors de la fusion des aspects deux à deux. Si d'autres critères s'ajoutent, l'adaptation devient beaucoup plus difficile à contrôler. Dans l'application vue ici, on peut vouloir par exemple ajouter comme critère le choix de l'utilisateur. Le problème de la fusion de trois aspects d'assemblage se pose alors. Il faut aussi considérer le nombre de cas de fusion qui augmente considérablement avec l'augmentation du nombre de critère.

En ce qui concerne la fusion de trois aspects d'assemblage, il serait judicieux de faire évoluer le langage d'écriture des aspects d'assemblages pour permettre la gestion de plusieurs niveaux de priorité d'application des aspects d'assemblage.

6.3.3 La mobilité de l'utilisateur

Le scenario mis en œuvre ici est le suivant : l'application est celle décrite figure 42, l'assemblage correspondant étant décrit figure 41. Il s'agit d'une application d'envoi de mails et de sauvegarde de notes. Cette application est hébergée sur un téléphone portable ou un PDA. L'utilisateur peut se trouver dans deux situations : hors de son véhicule ou dans son

véhicule. On considère que, hors de son véhicule, l'utilisateur peut utiliser l'application de manière classique, c'est à dire en remplissant les champs nécessaires à l'envoi d'un message ou à la sauvegarde d'une note en utilisant son PDA de manière classique, avec le clavier et éventuellement un stylet. On désire assurer la continuité du service fourni par l'application lorsque l'utilisateur est dans son véhicule. Dans ce cas, pour des raisons de sécurité, l'IHM du PDA n'est plus dans l'environnement d'exécution de l'application. Il faut donc utiliser un autre dispositif d'entrée-sortie pour que le service soit disponible. Par conséquent, un nouveau mode de fonctionnement de l'application est défini : le mode vocal.

Le mode vocal consiste à permettre l'utilisation de l'application par commandes vocales. Techniquement, deux mécanismes doivent être mis en place en commande vocale :

- le clic sur un bouton,
- la saisie dans une boîte de texte.

De plus, il faut penser à assurer la généricité de ces mécanismes. En d'autres termes, on doit pouvoir les appliquer à n'importe quelle application qui contient des boutons ou des boîtes de texte.

Le clic par commande vocale

Deux composants ont été développés pour cet usage. Le premier est un moteur de reconnaissance vocale en langue anglaise appelé "*VocalCommand*". L'API utilisée pour la reconnaissance vocale est le Microsoft Speech SDK 5.1. Des moteurs plus performants existent mais celui-ci est gratuit et suffisant pour l'usage que l'on va en faire.

Le code de ce composant contient une grammaire, c'est à dire une liste de mots clés ou de séquences clés. On peut au choix soit écrire les mots clés directement dans le code du composant ou faire en sorte qu'un fichier contenant tous les mots clés soit chargé à l'instanciation du composant. Ce dernier "écoute" l'utilisateur et est capable de réagir aux mots clés. Concrètement, un événement est émis à chaque fois qu'une séquence est reconnue. Le choix des mots clés est très important. D'abord pour l'utilisateur, les mots clés doivent être simples à retenir et intuitifs. Ils doivent également être suffisamment caractéristiques pour ne pas être confondus avec d'autres séquences de mots. Même si plusieurs composants "*VocalCommand*" apparaissent dans un assemblage, un seul moteur de reconnaissance vocale de type "*VocalCommand*" fonctionne. Le code de ce composant est visible en annexe 10.

Le second type de composant est appelé "*TextRecognition*". Sa fonction est de réagir à la reconnaissance d'un mot clé et un seul. Ce mot clé est contenu dans sa propriété "keyword". Lorqu'un mot clé est reconnu par un composant de type "*VocalCommand*", un événement est émis vers chaque composant de type "*TextRecognition*". Ces composants comparent la séquence reçue avec leurs mots clés respectifs. Le composant pour lequel la séquence et le mot clé sont identiques réagit, un événement est émis vers le bouton auquel le composant est lié et provoque un clic. Le code de ce composant est visible en annexe 11.

Enfin, un composant de type "*TextToSpeech*" a été utilisé pour la mise en place du clic en commande vocale. Il permet d'avoir un retour audio sur les actions effectuées.

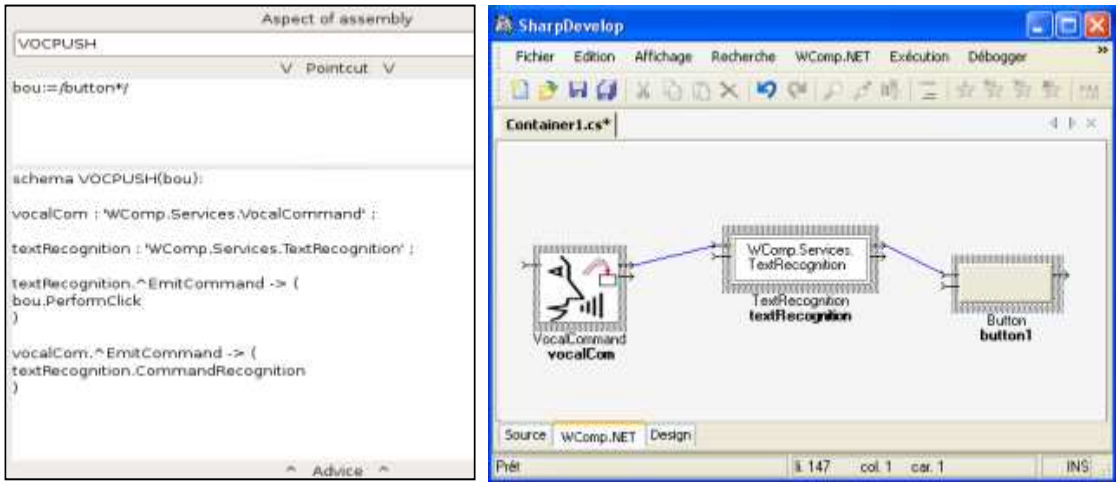


Figure 59 : **Application de l'aspect d'assemblage "VOCPUSH" (1)**

Sur la figure 59, on voit l'application d'un aspect mettant en place le clic par commande vocale sur un bouton. Le problème rencontré pour les contrats demeure. En effet, pour que l'application soit utilisable, il faut configurer le composant "*TextRecognition*" avec un mot clé. Il faut donc accéder aux propriétés de ce composant en écriture lors de son instanciation. Une autre difficulté réside dans le choix de ces mots clés. En effet, si le composant est configuré automatiquement lors de son instanciation, le choix de ce mot clé doit lui aussi être automatique. Comment le *AA designer* peut-il choisir des mots clés caractéristiques, qui seront connus intuitivement par l'utilisateur, quelle que soit l'application ?

Avant de tenter de répondre à cette question, il faut examiner un autre problème. En effet, lorsqu'on essaye d'appliquer l'aspect d'assemblage décrit sur la figure 59 à un assemblage contenant deux boutons, le résultat n'est pas celui escompté (voir figure 60).

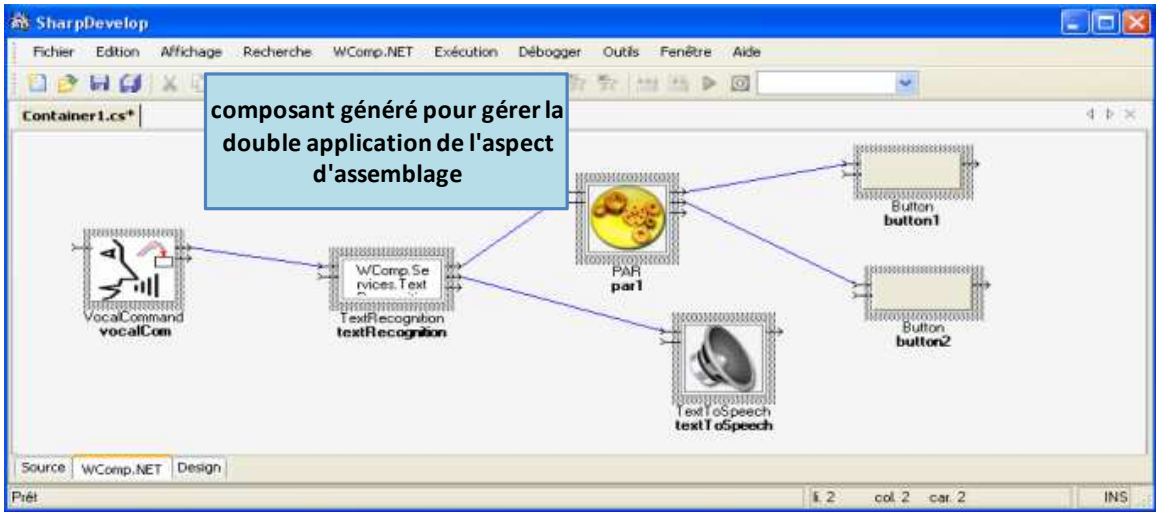


Figure 60 : **Application de l'aspect d'assemblage "VOCPUSH" (2)**

Grâce au point de coupe, l'aspect d'assemblage s'applique autant de fois qu'il y a de boutons dans l'assemblage. Néanmoins, les composants locaux ne sont instanciés qu'une fois. Or, si ce n'est pas gênant pour le composant "vocalCom" (car il n'y a qu'un seul moteur de reconnaissance vocale de type "*VocalCommand*") ou pour le composant "*textToSpeech*", cela pose un gros problème pour le composant "*textRecognition*". En effet, chaque bouton doit "posséder" son propre composant de type "*TextRecognition*", qui pourra reconnaître le

mot clé qui provoquera un clic. Cela prouve la nécessité d'instancier autant de jeux de composants locaux que d'applications de l'aspect d'assemblage. Cela ouvre la voie à d'autres problèmes. En effet, on ne sait pas à l'avance combien de fois un aspect d'assemblage va s'appliquer et pour l'instant, le langage ISL4WComp ne permet pas de créer plusieurs jeux de composants locaux. Alors si plusieurs composants locaux sont instanciés, comment les nommer ?

Les deux problèmes précédemment évoqués, c'est à dire le choix des mots clés et le nommage des composants locaux peuvent être résolus si le *AA designer* a la possibilité d'accéder en lecture aux noms des composants, donc plus généralement aux propriétés des composants et d'analyser la syntaxe de ces noms, pour en déduire les mots clés et les noms des composants locaux. Il est par conséquent nécessaire de définir une syntaxe des noms de composants, qui sera respectée lors de la conception des applications par assemblages. La liste des composants de l'application avec la syntaxe choisie pour leur nommage est donnée figure 60.

type du composant	usage du composant	paramètre de l'usage du composant	nom du composant
button	fill	**adress**	button_fill_adress
button	fill	**title**	button_fill_title
button	fill	**message**	button_fill_message
button	fill	**text**	button_fill_text
button	send	**mail**	button_send_mail
button	write	**note**	button_write_note
textBox	fill	adress	textBox_fill_adress
textBox	fill	title	textBox_fill_title
rtextBox	fill	message	rtextBox_fill_message
rtextBox	fill	text	rtextBox_fill_text
emitter	fill	adress	emitter_fill_adress
emitter	fill	title	emitter_fill_title
emitter	fill	text	emitter_fill_text
emitter	fill	message	emitter_fill_message
emitter	send	adress	emitter_send_adress
emitter	send	title	emitter_send_title
emitter	send	message	emitter_send_message
emitter	write	text	emitter_write_text
label	send	adress	label_send_adress
label	send	title	label_send_title
label	send	message	label_send_message
label	write	text	label_write_text
label	report		label_report
translator	fill	adress	translator_fill_adress
sendMail			sendMail
writeNote			writeNote

Figure 61 : **Liste et syntaxe des composants de l'application d'envoi de mail**

Le *AA designer* doit connaître cette syntaxe et être capable d'extraire des informations des noms des composants. Ainsi, la liste des mots clés est donnée par la liste des paramètres des usages des composants de type "*Button*". Il s'agit donc de "*adress*", "*title*", "*message*", "*text*", "*mail*" et "*note*". Le choix des mots clés est donc réglé. Néanmoins, ils ne sont pas très caractéristiques. En effet, un moteur de reconnaissance vocale en mode "dictée" va être mis en place par la suite. Dans ce cadre, ils pourraient aisément être confondus avec d'autres

mots. On décide d'ajouter le mot "*dot*" (en anglais, prononcer "dotte") avant et après chaque mot clé. Les séquences clés sont donc finalement "*dot adress dot*", "*dot title dot*", "*dot message dot*", "*dot text dot*", "*dot mail dot*" et "*dot note dot*".

Enfin, le *AA designer* va pouvoir générer des composants locaux avec des noms spécifiques au bouton sur lequel le clic vocal est mis en place. Par exemple, pour le bouton "*button_fill_adress*", il générera les composants "*vocalCommand_fill_adress*" et "*textRecognition_fill_adress*". Cette possibilité va se révéler très utile pour la suite. Sur la figure 62 est donné une partie de l'assemblage avec le clic en commande vocale mis en place.

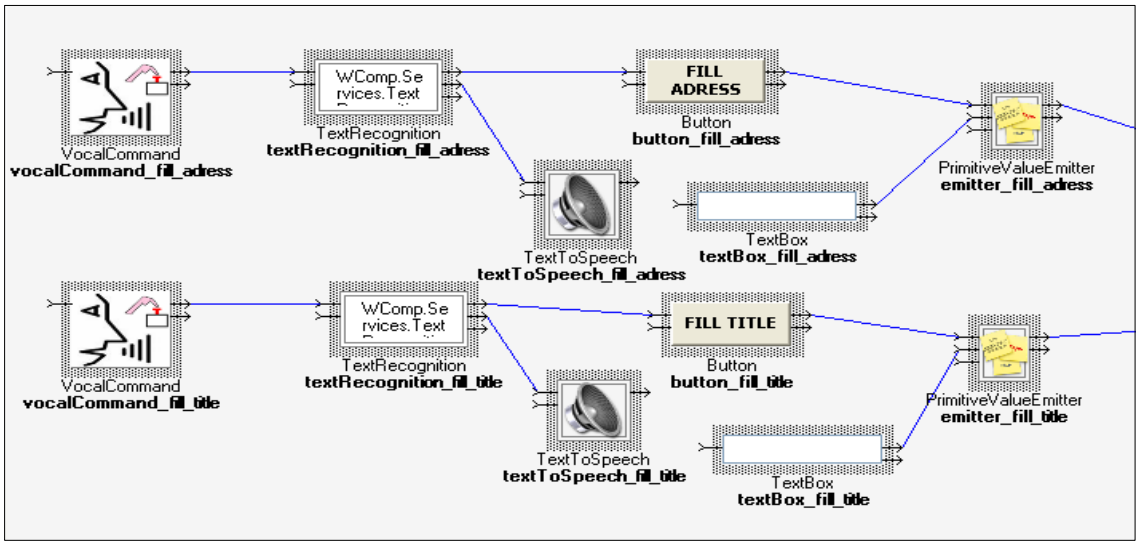


Figure 62 : **Assemblage "Clic en commande vocale"**

La saisie par commande vocale

La mise en place de ce mécanisme a nécessité le développement d'un composant dont le type est appelé "*SpeechToText*". Ce composant est similaire au composant de type "*VocalCommand*". Le différence réside d'une part dans la grammaire qui n'est pas constituée uniquement de quelques mots choisis mais de tous les mots de langue anglaise (du moins tous ceux que le moteur Microsoft connaît), et d'autre part par le fait que le moteur de reconnaissance vocale n'est pas démarré au moment de l'instanciation du composant. En effet, le moteur est déclenché dès qu'un mot clé est reconnu. Il faut alors gérer l'écoute du composant grâce à la propriété "*listenFlag*" du composant de type "*SpeechToText*", dont la valeur sera positionnée à "*true*" ou "*false*" selon les mots clés détectés. Le but d'un tel composant est de pouvoir dicter un texte qui s'affichera dans un composant de type "*textBox*".

Pour cet usage, le composant de type "*textRecognition*" présenté précédemment a été enrichi avec plusieurs évènements. Il ne permet plus seulement d'envoyer un événement à un bouton pour déclencher un clic, mais aussi d'envoyer des évènements sur un composant de type "*SpeechToText*" pour démarrer le moteur de reconnaissance vocale et mettre le composant "à l'écoute" et un événement sur un composant de type "*TextBox*" pour effacer son contenu. Voir en annexe 12 le code final du composant "*textRecognition*".

L'application de l'aspect VOCFILL_box donne le même problème que précédemment (voir fig.63). On suppose pour la suite que ce problème est résolu (voir fig. 64)

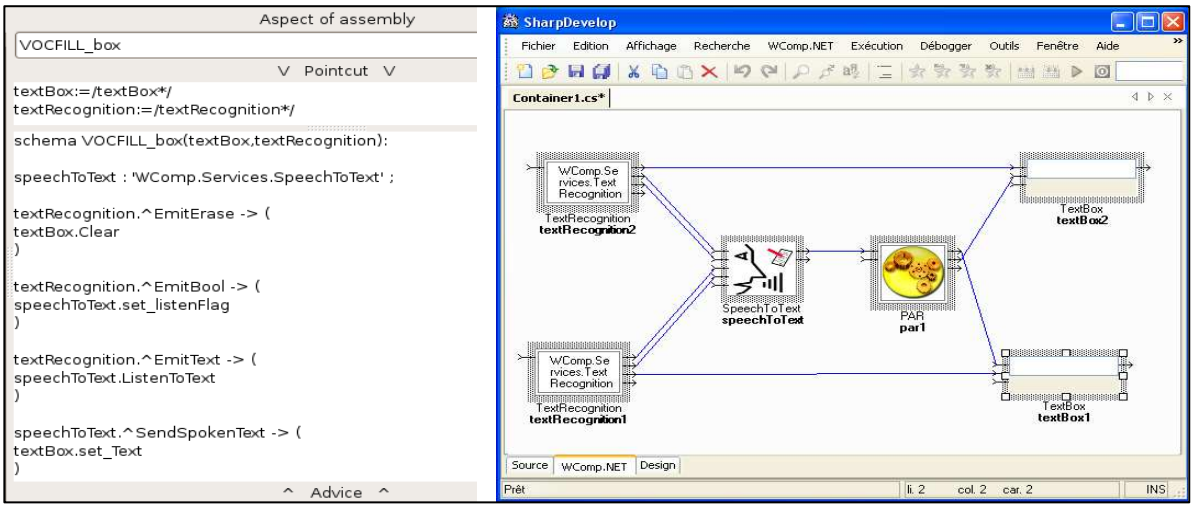


Figure 63 : **Application de l'aspect d'assemblage "VOCFILL_box" (1)**

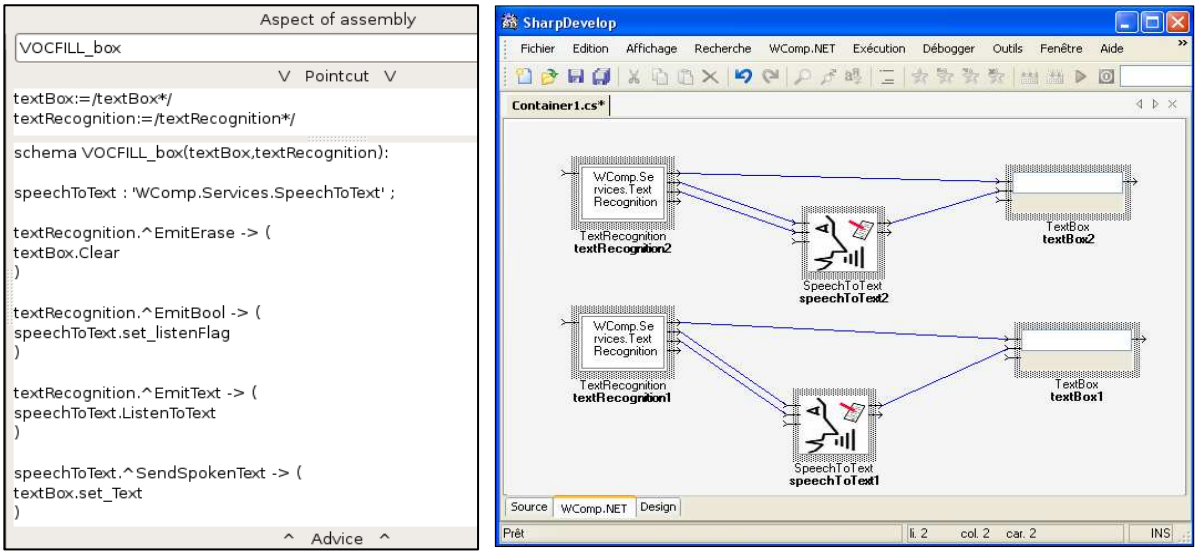


Figure 64 : **Application de l'aspect d'assemblage "VOCFILL_box" (2)**

Dans la suite, on utilise la notation "*textRecognition**" pour décrire des composants dont le nom commence par la chaîne de caractères "*textRecognition*" et se termine par n'importe quelle autre chaîne de caractères.

Un nouveau problème se pose alors. En effet, on désire l'application de l'aspect d'assemblage uniquement lorsqu'on a un composant "*textRecognition**" et un composant "*textBox**". Si le nombre de composants "*textRecognition**" est égal au nombre de composant "*textBox**", l'aspect d'assemblage s'applique. Or, dans l'exemple étudié ici, il y a cinq composants de type *textBox* et sept composants de type *textRecognition*. En effet, certains composants "textRecognition*" ne sont pas impliqués dans un mécanisme de saisie en commande vocale. L'aspect d'assemblage ne peut donc pas s'appliquer car il y a une incertitude sur l'endroit où il doit s'appliquer. Le *AA designer* ne peut pas déterminer quels sont les composants "*textRecognition**" concernés et ceux à mettre de côté. Si en revanche,

le nombre de composants "*textRecognition**" est égal au nombre de composants "*textBox**", l'aspect d'assemblage s'appliquera. Mais de quelle manière ?
Lorsqu'une paire de composants est impliquée dans l'application d'un aspect d'assemblage, le *AA designer* constitue toutes les paires qui seront concernées par ordre d'instanciation. Le premier des composants "*textRecognition**" instancié sera lié au premier des composants "*textBox**" instancié et ainsi de suite. Or, rien ne garantit que l'ordre d'instanciation convienne. Par exemple, le composant "*textRecogntion_fill_adress*" pourrait se trouver lié avec le composant "*textBox_fill_title*", ce qui rend l'application inutilisable.

Pour la résolution de ce dernier problème, le *AA designer* doit être capable de composer les paires (ou les trios, ou plus) de manière plus intelligente. La solution réside là encore dans la capacité du *AA designer* à exploiter des informations provenant des noms des composants, ces derniers respectant une syntaxe prédéfinie. On peut ajouter à la simple extraction de chaînes de caractères la possibilité de reconnaître des motifs en comparant des extraits de noms de composants. En annexe 13, l'assemblage de l'application en mode vocal.

6.3.4 Conclusion

Dans tous les exemples vus, l'application d'un aspect d'assemblage dépend d'une part de la sélection de l'aspect dans le *AA designer* et d'autre part de la présence de composants requis pour l'adaptation de l'application. En ce sens, le contexte n'est que partiellement pris en compte dans la plateforme WComp. La présence ou l'absence des dispositifs, ainsi que la sélection de l'aspect d'assemblage font partie du contexte de l'application. Mais qu'en-est-il du "quand ? ". En effet, il est nécessaire de pouvoir décider du moment où un aspect va s'appliquer ; plus précisément, il faut pouvoir empêcher son application même si les composants requis sont présents et si l'aspect d'assemblage est sélectionné. Dans ce sens, des travaux en cours actuellement visent à améliorer le *AA designer* en lui permettant de déclencher l'application d'un aspect d'assemblage sur la réception d'un évènement, émis par exemple par un capteur dans l'environnement. L'ajout de cette condition de déclenchement améliorera la prise en compte du contexte dans la plateforme WComp.

CONCLUSION

Dans un monde où les dispositifs d'informatique ambiante sont de plus en plus nombreux, les services ne sont plus liés à un seul dispositif, comme c'était le cas précédemment. Les services doivent être continuellement disponibles, quitte à changer de dispositif, et ce, à n'importe quel moment. Par exemple, le service messagerie électronique n'est plus enchaîné au PC traditionnel. Il a aujourd'hui entre autres supports certains téléphones portables qui permettent à un utilisateur d'accéder à ses mails dans de nouvelles situations. Une des étapes suivantes pourrait être de rendre ce service disponible dans un véhicule. Dans ce contexte, gérer l'hétérogénéité des dispositifs et assurer la continuité des services sont devenus des défis majeurs. Aussi, les recherches sur les intergiciels aidant à la conception d'applications adaptables à leur contexte sont nombreuses. Parmi les plateformes proposées, WComp présente plusieurs atouts. Tout d'abord, elle est une implémentation d'un modèle – le modèle SLCA – multi-paradigme, qui couvre l'ensemble des besoins de l'informatique ambiante, là où d'autres plateformes ne se concentrent que sur un ou deux paradigmes. En effet, la conception orientée composants, ainsi qu'une architecture de services améliorée pour prendre en compte l'interopérabilité, les communications évènementielles et la découverte dynamique répartie sont autant de concepts intégrés par le modèle SLCA et, par conséquent par son implémentation. Il faut remarquer que l'approche multi *designers* définie par SLCA est très intéressante, dans la mesure où elle permet une meilleure séparation des préoccupations. Un autre avantage réside dans le fait que la conception et l'adaptation des applications sont bien séparées. C'est le paradigme d'Aspect d'Assemblage qui prend en charge l'adaptation, tandis que le modèle SLCA régit la conception, ce qui permet une réutilisabilité des aspects d'assemblage. Enfin, et ce n'est pas négligeable, la plateforme WComp est facile à installer et à utiliser. En effet, peu de ressources systèmes sont nécessaires, et l'installation des logiciels nécessaires ne prend que peu de temps.

La plateforme WComp est un très bon outil de conception d'application sous forme d'assemblage de composants, malgré des problèmes de synchronisation à prendre en compte. Le designer d'aspect d'assemblage, quant à lui, doit subir quelques évolutions pour permettre une meilleure prise en compte du contexte et des possibilités d'adaptation accrues. Il s'agit assurément d'une excellente voie de recherche car les applications devront pouvoir s'adapter à leur contexte dans l'informatique de demain.

GLOSSAIRE

***Autonomic computing* (ou *resource-aware computing*)** : adaptation d'une application informatique à son contexte de manière autonome, c'est-à-dire sans intervention de l'utilisateur.

Bluetooth : Spécification de l'industrie des télécommunications. Elle utilise une technologie radio courte distance destinée à simplifier les connexions entre les appareils électroniques. Elle a été conçue dans le but de remplacer les câbles entre les ordinateurs et les imprimantes, les scanners, les claviers, les souris, les téléphones portables, les PDA, les autoradios et les appareils photo numériques.

C.C.M. : Corba Component Model : modèle de composants pour l'architecture Corba.

Corba : acronyme de Common Object Request Broker Architecture, Corba est une architecture logicielle, pour le développement de composants et d'Object Request Broker ou ORB. Ces composants, qui sont assemblés afin de construire des applications complètes, peuvent être écrits dans des langages de programmation distincts, être exécutés dans des processus séparés, voire être déployés sur des machines distinctes. Corba est un standard maintenu par l'Object Management Group.

Domotique : La domotique regroupe l'ensemble des techniques et technologies permettant de superviser, d'automatiser, de programmer et de coordonner les tâches de confort, de sécurité, de maintenance et plus généralement de services dans l'habitat individuel ou collectif.

E.J.B. : La technologie Enterprise JavaBeans (EJB) est une architecture de composants logiciels côté serveur pour la plateforme de développement J2EE.

End-user programming : programmation pour utilisateur final.

Environnement d'exécution: sous-ensemble de l'infrastructure logicielle, ensemble des ressources disponibles du système à un instant donné.

Fractal : Fractal est un modèle de composant modulaire et extensible qui peut être utilisé pour concevoir, implanter, déployer et reconfigurer les systèmes et applications, du système d'exploitation à la plateforme "middleware" et à l'interface graphique. Le but de Fractal est de réduire les coûts de développement, de déploiement et de maintenance des logiciels en général.

Framework : en informatique, un *framework* est un espace de travail modulaire. C'est un ensemble de bibliothèques, d'outils et de conventions permettant le développement d'applications. Il fournit suffisamment de briques logicielles et impose suffisamment de rigueur pour pouvoir produire une application aboutie et facile à maintenir. Ces composants sont organisés pour être utilisés en interaction les uns avec les autres.

Fscript : Langage de script orienté objet basé sur Smalltalk et développé par Philippe Mougin. Smalltalk est un langage de programmation objet.

G.E.N.A. : *General Event Notification Architecture,* mécanisme de notification/souscription mis en œuvre dans UPnP, permettant l'envoi d'événement d'un dispositif à un point de contrôle.

H.T.T.P. : Le *HyperText Transfer Protocol*, littéralement le "protocole de transfert hypertexte", est un protocole de communication client-serveur développé pour le World Wide Web.

I.D.L. : Interface Description Language (appelé aussi interface definition language) : langage voué à la définition de l'interface de composants logiciels, laquelle permet de faire communiquer les modules implémentés dans des langages différents.

I.H.M. : Interface Homme-Machine. Dispositif d'interaction entre l'humain et l'ordinateur.

Infrastructure logicielle : ensemble des équipements avec lesquels l'application peut interagir.

Intergiciel ou *middleware* : En informatique, un intergiciel (en anglais *middleware*) est un logiciel servant d'intermédiaire de communication entre plusieurs applications, généralement complexes ou distribuées sur un réseau informatique. Le terme *middleware* vient de l'anglais *middle* (du milieu) et *software* (logiciel). Diverses francisations ont été proposées et intergiciel semble le terme le plus répandu.

JavaBeans : Les JavaBeans sont des composants logiciels écrits en langage Java. La spécification JavaBeans de Sun Microsystems définit les JavaBeans comme « des composants logiciels réutilisables manipulables visuellement dans un outil de conception ». En dépit de quelques similarités, les JavaBeans ne doivent pas être confondus avec les Entreprise JavaBeans (EJB), une technologie de composants côté serveur faisant partie de J2EE (Java2 Entreprise Edition).

JINI : A l'origine développé par Sun Microsystems, Jini est une architecture réseau pour la construction de systèmes distribués dans la forme de services coopérants modulaires. Cette technologie libère les ordinateurs de toute dépendance à l'égard des systèmes d'exploitation. En considérant les périphériques et les logiciels comme des objets indépendants qui peuvent communiquer, Jini peut les réunir en fédération d'objets qui s'installent automatiquement et qui fonctionnent dès qu'ils sont branchés. Jini n’est pas un acronyme.

L.D.A.P. : *Lightweight Directory Access Protocol* est à l'origine un protocole permettant l'interrogation et la modification des services d'annuaire. Ce protocole repose sur TCP/IP. Il a évolué pour représenter une norme pour les systèmes d'annuaires.

Metadonnées : donnée servant à définir ou décrire une autre donnée.

Métamodèle : modèle d'un langage de description de modèles.

.NET components : modèle de composnats pour l’architecture .NET de microsoft.

.NET delegate : Les "delegate" sont une forme de pointeur de fonction utilisée par le. NET Framework. Ils spécifient une méthode à invoquer et éventuellement un objet sur lequel invoquer cette méthode. Ils sont utilisés, entre autres choses, à mettre en œuvre des rétro appels et des observateurs d'évènements.

Ontologie : système de représentation des connaissances. En informatique, une ontologie est l'ensemble structuré des termes et concepts fondant le sens d'un champ d'informations.

O.S.G.I. : *L’OSGi Alliance* (précédemment connue sous le nom de *Open Services Gateway Initiative*) est organisation Open fondée en mars 1999. L’Alliance et ses membres ont spécifié une plateforme de services basée sur le langage Java qui peut être gérée de manière distante. Le *framework* implémente un modèle de composants dynamique et complet, comblant un manque dans les environnements Java/VM traditionnels

O.W.L. : acronyme réarrangé de *Web Ontology Language*, "langage d'ontologie web", langage informatique utilisé pour modéliser des ontologies, ensemble de concepts et connaissances.

Paradigme : Un paradigme de programmation est un style fondamental de programmation informatique qui traite de la manière dont les solutions aux problèmes doivent être formulées dans un langage de programmation.

P.D.A. : Personal Digital Assistant. Un assistant personnel ou ordinateur de poche est un appareil numérique portable.

Pervasif : Ce terme n'est pas reconnu officiellement dans l'usage en langue française. En informatique, c'est un adjectif exprimant la diffusion à travers toutes les parties d'un système d'information.

Proxy : serveur informatique qui a pour fonction de relayer des requêtes entre un poste client et un serveur.

Radio-fréquence (RF) : mode de communication par ondes radioélectriques. Ces ondes ont une fréquence comprise entre 9 kHz et 3000 GHz.

Réalité augmentée : Par un système de réalité augmentée on entend un système informatique qui rend possible de superposer l'image d'un modèle virtuel 3D ou 2D sur une image de la réalité et ceci en temps réel. Le concept de réalité augmentée vise donc à accroître notre perception du monde réel, en y ajoutant des éléments fictifs, non visibles a priori.

Réflexivité : La réflexivité logicielle est un concept lié à la notion de tolérance aux fautes. Cette dernière consiste, pour un système, à réagir aux états erronés rapidement, de manière à empêcher que ces erreurs ne conduisent à un dysfonctionnement visible pour l'utilisateur.

R.F.I.D. : radio frequency identification soit radio-identification en français. Méthode pour stocker et récupérer des données à distance en utilisant des marqueurs appelés "radio-étiquettes" ("RFID tag" en anglais).

R.P.C. : **R**emote **P**rocedure **C**all : protocole permettant de faire des appels de procédures sur un ordinateur distant à l'aide d'un serveur d'application. Ce protocole est utilisé dans le modèle client-serveur et permet de gérer les différents messages entre ces entités.

R.S.S. : RSS désigne une famille de formats XML utilisés pour la syndication de contenu Web. Ce standard est habituellement utilisé pour obtenir les mises à jour d'information dont la nature change fréquemment, typiquement cela peut être des listes de tâches dans un projet, des prix, des alertes de toute nature, des nouveaux emplois proposés, les sites d'information ou les blogs.

S.L.P. : Service Location Protocol : protocole de découverte de services permettant à des ordinateurs et à d'autres dispositifs de découvrir des services dans un LAN sans configuration préalable. SLP a été conçu pour les petits réseaux sans gestion aussi bien que pour les gros réseaux d'entreprise. Il est défini dans la RFC 2608.

S.M.S. : Short Message Service. Service de messages courts des téléphones portables.

S.O.A. : Une architecture orientée services (Services Oriented Architecture) est une architecture logicielle s'appuyant sur un ensemble de services simples. L'objectif d'une architecture orientée services est donc de décomposer une fonctionnalité en un ensemble de

fonctions basiques, appelées services, fournies par des composants et de décrire finement le schéma d'interaction entre ces services.

S.O.A.P. : (acronyme de *Simple Object Access Protocol*) ; protocole de RPC orienté objet bâti sur XML. Il permet la transmission de messages entre objets distants, ce qui veut dire qu'il autorise un objet à invoquer des méthodes d'objets physiquement situés sur un autre serveur.

T.C.P./I.P. : Le *Transmission Control Protocol* (protocole de contrôle de transmissions"), est un protocole de transport fiable, en mode connecté, documenté dans la RFC 793 de l' IETF (Internet Engineering Task Force) ; L'*Internet Protocol*, généralement abrégé IP, est un protocole de communication de réseau informatique. IP est le protocole d'Internet.

Technologie COM : technologie Microsoft (Component Object Model) permettant à des composants logiciels de communiquer entre eux. COM est utilisée par les développeurs pour créer des composants logiciels réutilisables, pour lier ces composants entre eux pour créer des applications, et tirer avantage des services Windows.

Tissage : dans le cadre des langages d'aspects, il s'agit d'un entrelacement du code. On parle aussi de tissage d'aspects d'assemblage, qui correspond à l'application de plusieurs aspects d'assemblage à un ensemble de composants.

Ubiquitaire : En informatique, le terme ubiquitaire désigne un environnement d'intelligence artificielle dans lequel les ordinateurs et réseaux sont "enfouis" et "intégrés" dans le monde réel. L'utilisateur a accès à un ensemble de services au travers d'interfaces distribuées se voulant intelligentes, dont il est entouré. Ces interfaces s'appuient sur des technologies intégrées dans les objets familiers.

U.D.D.I. : acronyme de *Universal Description Discovery and Integration* **:** annuaire de services basé sur XML et plus particulièrement destiné aux services Web.

U.R.L. : Uniform Ressource Locator. Chaîne de caractères ASCII utilisée pour adresser les ressources sur un réseau.

Utilisabilité : il semble que ce terme ne soit pas reconnu dans tous les dictionnaires. Il s'agit de la capacité d'un système à permettre à ses utilisateurs normaux de faire efficacement ce pour quoi ils l'utilisent.

W3C : Le *World Wide Web Consortium*, abrégé par le sigle W3C, est un organisme de normalisation fondé en octobre 1994 comme un consortium chargé de promouvoir la compatibilité des technologies du *World Wide Web*.

WI-Fi 802.11b : technologie de réseau informatique sans fil mise en place pour fonctionner en réseau interne et, depuis, devenue un moyen d'accès à haut débit à Internet. elle est basée sur la norme IEEE 802.11.

W.S.D.L. : *Web Services Description Language*, en français, langage de description de Service Web*.* Ce langage sert à décrire une interface publique d'accès à un Service Web*,* notamment dans le cadre d'architectures de type SOA.

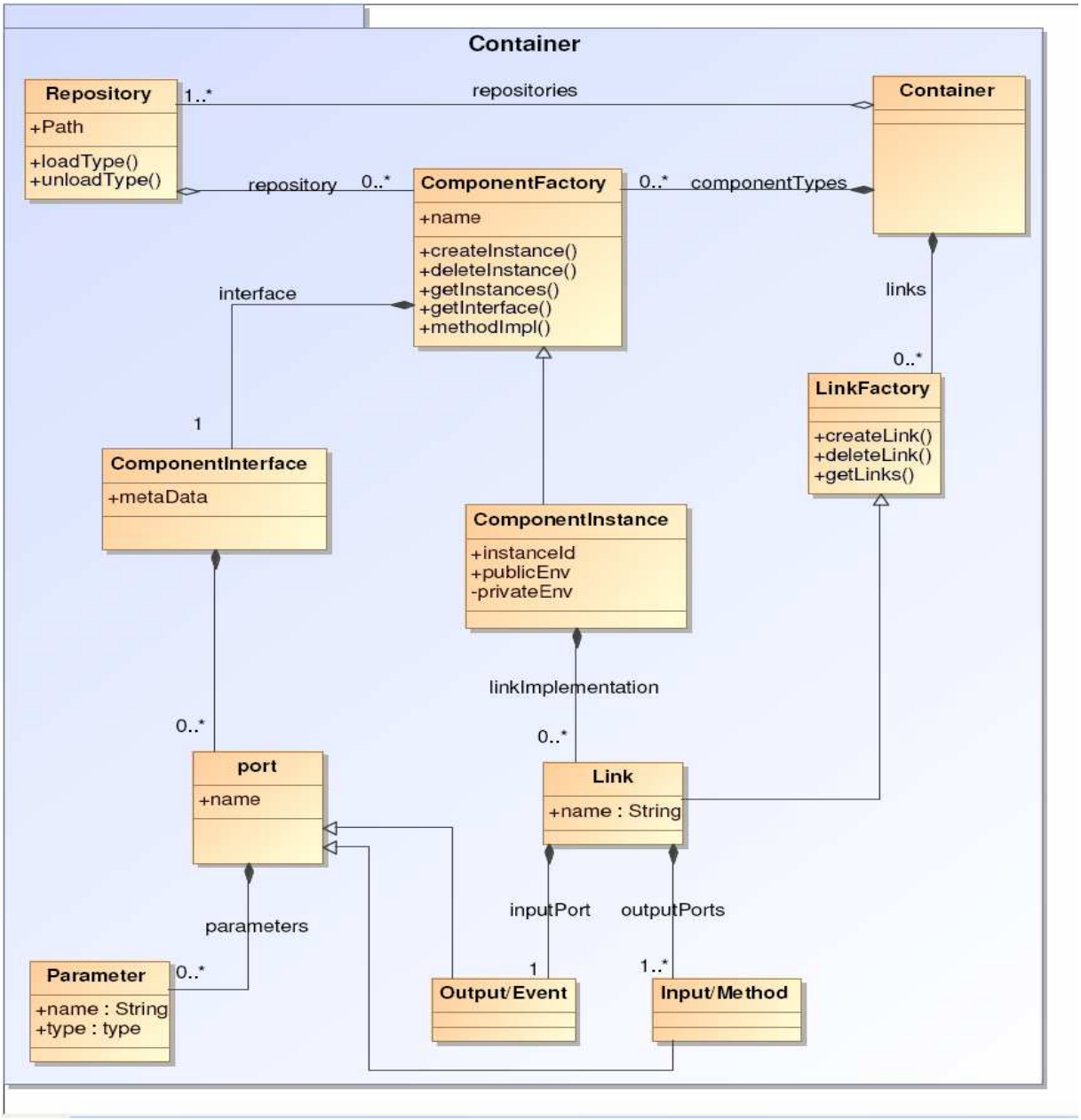
X.M.L. : *eXtensible Markup Language*, ("langage de balisage extensible") est un langage informatique de balisage générique. Son objectif initial est de faciliter l'échange automatisé de contenus entre systèmes d'informations hétérogènes (interopérabilité).

ANNEXES

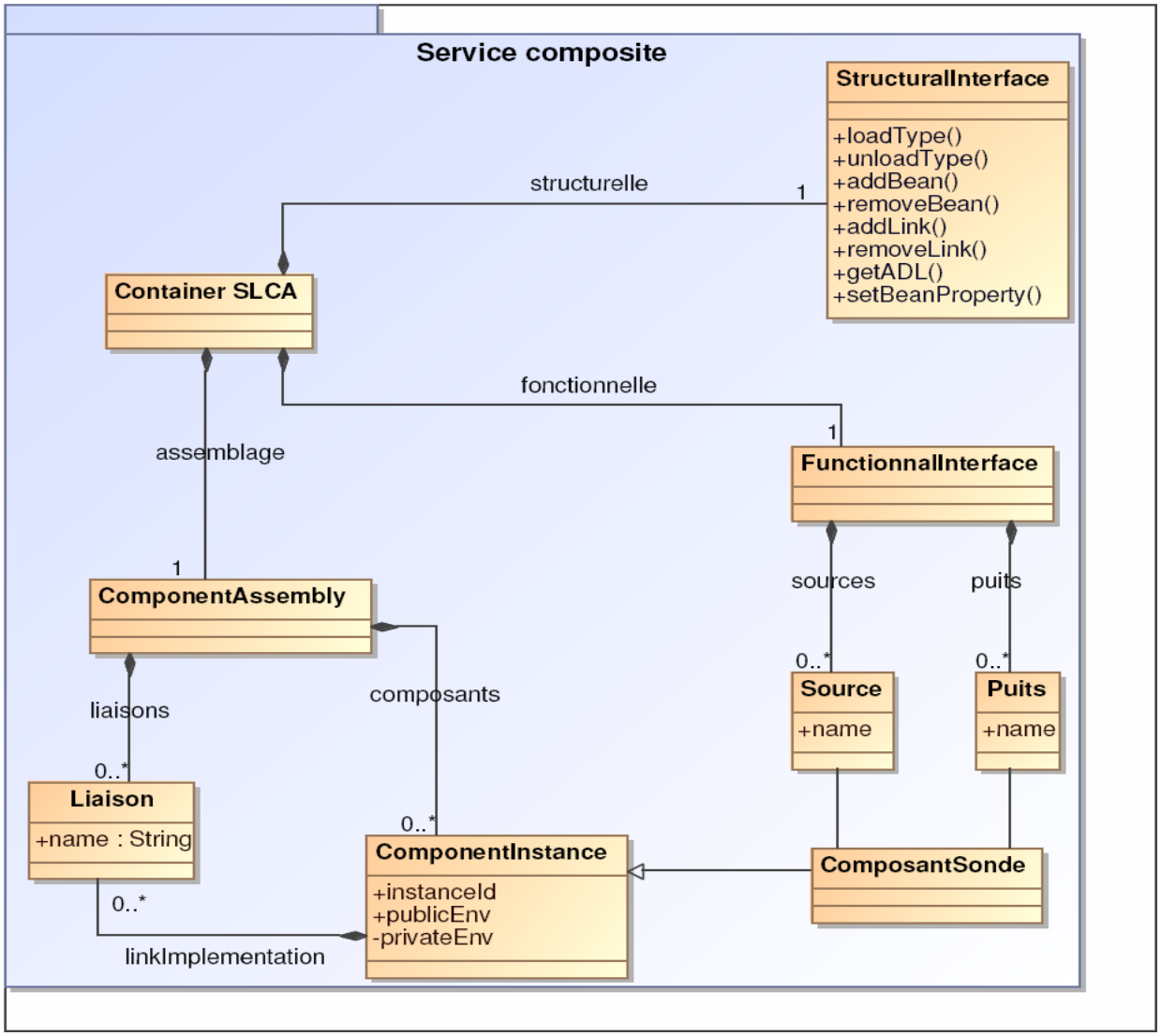
Annexe 1 : liste des centres de recherche utilisant des intergiciels sensibles au contexte

Intergiciel	Organisme(s) de recherche
GAIA	University of Illinois at Urbana-Champaign Digital Computer Lab USA
RSCM	Arizona State University département informatique et science de l'ingénieur USA
CARMEN	Dynamic Interactions Group du Media Lab Europe (MIT) Distributed Systems Group du Trinity College Dublin Europe-USA 2002-2005
SOCAM	Université de Singapore & Instute for infocomm Research Singapore
Amigo	16 sociétés européennes, institut de recherche et universités (Philips,FT R & D ...) Europe
CORTEX	Universidade de Lisboa(Portugal) Lancaster University(United Kingdom) Trinity College (Ireland) Universität Ulm(Germany) Europe
Aura	Carnegie Mellon University Département informatique USA
CAMiDO	GET/INT, CNRS UMR Samovar Evry France
CARISMA	University College. London Département informatique England
Oxygen	MIT Project Oxygen Computer Science and Artificial Intelligence Laboratory USA
SAFRAN	Projet Obasco Ecoles des mines de Nantes FT R&D INRIA Siemens AG (Munich) VirtualLogix France
SCaLaDE	Université de Bologne & université de Ferrara Italie

Annexe 2 : Meta-modèle de LCA : composants légers



Annexe 3 : Meta-modèle de SLCA interfaces des services composites



Annexe 4 : Modèle de code pour un bean fourni par la plate-forme WCOMP

SharpDevelop

Fichier Edition Affichage Recherche Exécution Débogger Outils Fenêtre Aide

Bean1.cs

```
/*
 * Created by SharpDevelop.
 * User: Eric
 * Date: 20/05/2008
 * Time: 11:56
 *
 * To change this template use Tools | Options | Coding | Edit Standard Headers.
 */

using System;
using WComp.Beans;
using WComp.EventedBeans;

namespace WComp.Beans
{
    /// <summary>
    /// Description of Bean1.
    /// This is a sample bean, which has an integer evented property.
    /// </summary>
    [Bean]
    public class Bean1 : EventedDrawable
    {
        /// <summary>
        /// Fill in private attributes here.
        /// </summary>
        private int property;


        /// <summary>
        /// This property will appear in bean's property panel and bean's input functions.
        /// </summary>
        public int MyProperty
        {
            get
            {
                return property;
            }
            set
            {
                property = value;
                FireIntEvent(property);     // event will be fired for every property set.
            }
        }

        /// <summary>
        /// Here are the delegate and his event.
        /// A function checking nullity should be used to fire events (like FireIntEvent).
        /// </summary>
        public delegate void IntValueEventHandler(int val);
        public event IntValueEventHandler PropertyChanged;

        private void FireIntEvent(int i){
            if (PropertyChanged != null)
            {
                PropertyChanged(i);
            }
        }
    }
}
```

Prêt li. 2 col. 2 car. 2 INS

Annexe 5 : Code en C# du bean AssemblyHandler

```
using System;
using WComp.Beans;
using WComp.EventedBeans;

namespace WComp.Beans
{
	[Bean]

	public class AssemblyHandler : EventedDrawable
	{
		public delegate void StringValueEventHandler(string val);
		public event StringValueEventHandler removeBeanEvent, removeLinkEvent;

		public delegate void String2ValueEventHandler(string val1, string val2);
		public event String2ValueEventHandler addBeanEvent;

		public delegate void String3ValueEventHandler(string val1, string val2, string val3);
		public event String3ValueEventHandler changeBeanPropertyEvent;

		public delegate void String5ValueEventHandler(string val1, string val2, string val3, string val4, string val5);
		public event String5ValueEventHandler addLinkEvent;

		private void FireRemoveBeanEvent(string a){
			if (removeBeanEvent != null)
			{
				removeBeanEvent(a);
			}
		}
		private void FireRemoveLinkEvent(string a){
			if (removeLinkEvent != null)
			{
				removeLinkEvent(a);
			}
		}
		private void FireAddBeanEvent(string a, string b){
			if (addBeanEvent != null)
			{
				addBeanEvent(a,b);
			}
		}
		private void FireAddLinkEvent(string a, string b, string c, string d, string e){
			if (addLinkEvent != null)
			{
				addLinkEvent(a,b,c,d,e);
			}
		}
		private void FireChangeBeanPropertyEvent(string a, string b, string c){
			if (changeBeanPropertyEvent != null)
			{
				changeBeanPropertyEvent(a,b,c);
			}
		}

		public void changeItem(string str)
		{
			string input ;
			string[] output = null;
			input = str;
			output = input.Split(' ');

			if (output.Length == 3 && output[0]=="addBean"){
				FireAddBeanEvent(output[1],output[2]);
			} else if (output.Length == 5 && output[0]=="addLink"){
				FireAddLinkEvent(output[1],output[2],output[3],output[4],"");
			} else if (output.Length == 6 && output[0]=="addLink"){
				FireAddLinkEvent(output[1],output[2],output[3],output[4],output[5]);
			} else if (output.Length == 2 && output[0]=="removeLink"){
				FireRemoveLinkEvent(output[1]);
			} else if (output.Length == 2 && output[0]=="removeBean"){
				FireRemoveBeanEvent(output[1]);
			} else if (output.Length == 4 && output[0] =="changeBeanProperty"){
				FireChangeBeanPropertyEvent(output[1],output[2],output[3]);
			}
		}

	}
}
```

Annexe 6 : Code en C# du bean BoolToString

```
using System;
using WComp.Beans;
using WComp.EventedBeans;

namespace WComp.InterfaceTranslator
{
	[Bean(Category="Interface Translator")]
	public class BoolToString : EventedDrawable
	{
		private bool lastVal;


		public bool lastValue
		{
			get
			{
				return lastVal;
			}
			set
			{
				lastVal = value;
			}
		}

		public void Convert(bool b){

			if (b == true)
			{
				FireStringEventHandler("true");
				lastVal = true;
			}
			else if (b == false)
			{
				FireStringEventHandler("false");
				lastVal = false;
			}

		}

		public delegate void StringEventHandler(string s);
		public event StringEventHandler StringEvent;

		private void FireStringEventHandler(string s){
			if (StringEvent != null)
				StringEvent(s);
		}
	}
}
```

Annexe 7 : Code en C# du composant stringTranslator

```
using System;
using System.IO;
using System.Text;
using WComp.Beans;
using WComp.EventedBeans;

namespace WComp.InterfaceTranslator
{
	[Bean(Category="Interface Translator")]
	public class StringTranslator2 : EventedDrawable
	{
		private string[] input = new string[] { };
		private string[] output = new string[] { };

		#region properties
		public string[] InputStrings{
			get{	return input;		}
			set{	input = value;		}
		}

		public string[] OuputStrings{
			get{	return output;		}
			set{	output = value;		}
		}

		public string fichierConfig{
			get{	return fichierConfig;			}
			set{	fichierConfig = value;	}
		}
		#endregion
		public delegate void TranslatedEventHandler(string val);
		public event TranslatedEventHandler Translated;
		public void Translate(string str){
			if (Translated != null && str != null)
			{
				loadInputOutput(fichierConfig);
				Translated(DoTranslation(str));
			}
		}
		private void loadInputOutput(string fic){
			if ((input.Length != 0) && (output.Length != 0) && (fic != null)){
				FileInfo fi = new FileInfo(fic);
				if (fi.Exists)  // vérification de l'existence du fichier
				{
					string s;
					int i = 0;
					int j = 0;
					FileStream fs = new FileStream(fic,FileMode.Open); // ouverture du fichier de config
					StreamReader sr = new StreamReader(fs);
					do{
						s = sr.ReadLine(); // lecture de la ligne
						if (s != null){
							input[i] = s;
							i = i + 1;
						}
					} while (s != null); // on continue jusqu'à trouver la première ligne vide
					s = sr.ReadLine();
					do{
						s = sr.ReadLine(); // lecture de la ligne
						input[j] = s;
						j = j + 1;
					} while (j < i);
					sr.Close();
					fs.Close();
				}
			}

		}
		private string DoTranslation(string str)
		{
			// !! Array.IndexOf(input, str) does not work since
			// !! it is not impelemented in .NET Compact Framework
			int idx = Array.IndexOf(input, str, 0, input.Length);
			if (idx >= 0 && idx < output.Length)
			{
				str = output[idx];
			}
			return str;
		}
	}
}
```

Annexe 8 : Code en C# du composant sendMail

```
using System;
using System.Web.Mail;
using WComp.Beans;
using WComp.EventedBeans;

namespace WComp.Beans
{
	[Bean(Category="MessageSender")]
	public class SendMail : EventedDrawable
	{
		private string EMailTo, MessageTitle, AdressFrom, ServerSMTP;
		private bool HTMLChoice;

		public string EMailFrom{
			get{return AdressFrom;}
			set{AdressFrom = value;}
		}
		public bool inHTML{
			get{return HTMLChoice;}
			set{HTMLChoice = value;}
		}
		public string SMTPServer{
			get{return ServerSMTP;}
			set{ServerSMTP = value;}
		}
		public void UpdateAdress(string adress){
			EMailTo = adress;
		}
		public void UpdateMessageTitle(string title){
			MessageTitle = title;
		}
		public void SendMail(string message){
			string Retour = string.Empty;
			MailMessage msg = null;
			System.Text.Encoding MyEncoding = System.Text.Encoding.GetEncoding("iso-8859-1");
			try {
				msg = new MailMessage();
				msg.Body = message;
				msg.BodyEncoding = MyEncoding;
				if (InHTML)
				{ msg.BodyFormat = MailFormat.Html; }
				else
				{ msg.BodyFormat = MailFormat.Text; }
				msg.Subject = MessageTitle;
				msg.From = EMailFrom;
				msg.To = EMailTo;
				SmtpMail.SmtpServer = SMTPServer;
				SmtpMail.Send(msg);
				Retour = "Mail sent to "+ MailTo;
				FireDisplayReport(Retour);
			}
			catch(Exception ex) {
				Retour = "Error in Sendmail function - Details : "+ ex.ToString();
				FireDisplayReport(Retour);
			}
			finally {
				msg = null;
				MyEncoding = null;
			}
		}

		public delegate void StringValueEventHandler(string val);
		public event StringValueEventHandler PropertyChanged,DisplayReport;

		private void FireStringEvent(string s){
			if (PropertyChanged != null)
			{PropertyChanged(s);}
		}

		private void FireDisplayReport(string s){
			if (DisplayReport != null)
			{DisplayReport(s);}
		}
	}
}
```

Annexe 9 : Code en C# du composant WriteNote

```
using System;
using System.IO;
using WComp.Beans;
using WComp.EventedBeans;


namespace WComp.Services
{
	[Bean(Category="Services")]

	public class WriteNote : EventedDrawable
	{
		private string path;
		private string format;

		public string NoteDirectory
		{
			get{return path;}
			set{path = value;}
		}

		public string FileFormat
		{
			get{return format;}
			set{format = value;}
		}

		public void SaveNote(string note)
		{
			Directory.CreateDirectory(NoteDirectory);
			DateTime CurrTime = DateTime.Now;
			string year = CurrTime.Year.ToString();
			string month = CurrTime.Month.ToString();
			string day = CurrTime.Day.ToString();
			string hour = CurrTime.Hour.ToString();
			string min = CurrTime.Minute.ToString();


			string FilePath = NoteDirectory + year + month + day + "_" + hour + min + "." + FileFormat;

			StreamWriter fic = File.CreateText(FilePath);
			fic.Write(note);
			fic.Close();

		}

		/// <summary>
		/// Here are the delegate and his event.
		/// A function checking nullity should be used to fire events (like FireIntEvent).
		/// </summary>
		public delegate void StringValueEventHandler(string val);
		public event StringValueEventHandler NoteDirectoryChanged, FileFormatChanged;

		private void FireNoteDirectoryEvent(string s){
			if (NoteDirectoryChanged != null)
			{
				NoteDirectoryChanged(s);
			}
		}

		private void FireFileFormatEvent(string s){
			if (FileFormatChanged != null)
			{
				FileFormatChanged(s);
			}
		}
	}
}
```

Annexe 10 : Code en C# du composant VocalCommand

```
using System;
using System.Text;
using System.IO;
using WComp.Beans;
using WComp.EventedBeans;
using SpeechLib;

namespace WComp.Services
{
	[Bean(Category="Services")]

	public class VocalCommand : EventedDrawable
	{
	// SR engine
	private SpeechLib.SpSharedRecoContext RecognitionContext;

	// what the engine will recognize : a grammar
	private ISpeechRecoGrammar Grammar;

	// grammar rule
	private SpeechLib.ISpeechGrammarRule CommandsRule;

	// constructeur
	public VocalCommand(){
		init_SREngine();
	}

	private void init_SREngine()	{
	// instance of SR engine
	RecognitionContext = RecoContext.GetInstance();
	// instance of grammar and init of grammar
	Grammar = RecognitionContext.CreateGrammar(0);
	Grammar.DictationLoad("",SpeechLoadOption.SLOStatic);
	// only for dictation mode
	// Grammar.DictationSetState(SpeechRuleState.SGDSActive);

	// setup of grammar rule : here, only one rule for this grammar
	CommandsRule = Grammar.Rules.Add("CommandsRule", SpeechRuleAttributes.SRATopLevel | SpeechRuleAttributes.SRADynamic, 1);
		object dummy = 0;
	CommandsRule.InitialState.AddWordTransition(null, "dot adress dot"," ",SpeechGrammarWordType.SGLexical,null,0, ref dummy,0);
	CommandsRule.InitialState.AddWordTransition(null, "dot title dot"," ",SpeechGrammarWordType.SGLexical,null,0, ref dummy,0);
	CommandsRule.InitialState.AddWordTransition(null, "dot message dot"," ",SpeechGrammarWordType.SGLexical,null,0, ref dummy,0);
	CommandsRule.InitialState.AddWordTransition(null, "dot text dot"," ",SpeechGrammarWordType.SGLexical,null,0, ref dummy,0);
	CommandsRule.InitialState.AddWordTransition(null, "dot mail dot"," ",SpeechGrammarWordType.SGLexical,null,0, ref dummy,0);
	CommandsRule.InitialState.AddWordTransition(null, "dot note dot"," ",SpeechGrammarWordType.SGLexical,null,0, ref dummy,0);
	CommandsRule.InitialState.AddWordTransition(null, "dot erase dot"," ",SpeechGrammarWordType.SGLexical,null,0, ref dummy,0);

	Grammar.Rules.Commit();
	// start the SR engine
	Grammar.CmdSetRuleState("CommandsRule", SpeechRuleState.SGDSActive);
	// turns off the automatic learning mode because it's a command recognition application
	RecognitionContext.Recognizer.SetPropertyNumber("AdaptationOn",0);
	// This event delegate will be called whenever a phrase is recognized.
	RecognitionContext.Recognition += new SpeechLib._ISpeechRecoContextEvents_RecognitionEventHandler

	(this.RecoContext_Recognition);
	}

	private void RecoContext_Recognition(int StreamNumber,object StreamPosition,
		SpeechLib.SpeechRecognitionType RecognitionType,SpeechLib.ISpeechRecoResult Result)
	{
		string command = Result.PhraseInfo.GetText(0,-1,true);
		// envoi de la commande vers un composant textRecognitionEmitter
		FireCommandEvent(command);
	}

	public delegate void StringValueEventHandler(string str);
	public event StringValueEventHandler EmitCommand;

	private void FireCommandEvent(string s){
		if (EmitCommand != null)
		{
			EmitCommand(s);
		}
	}
}
}
```

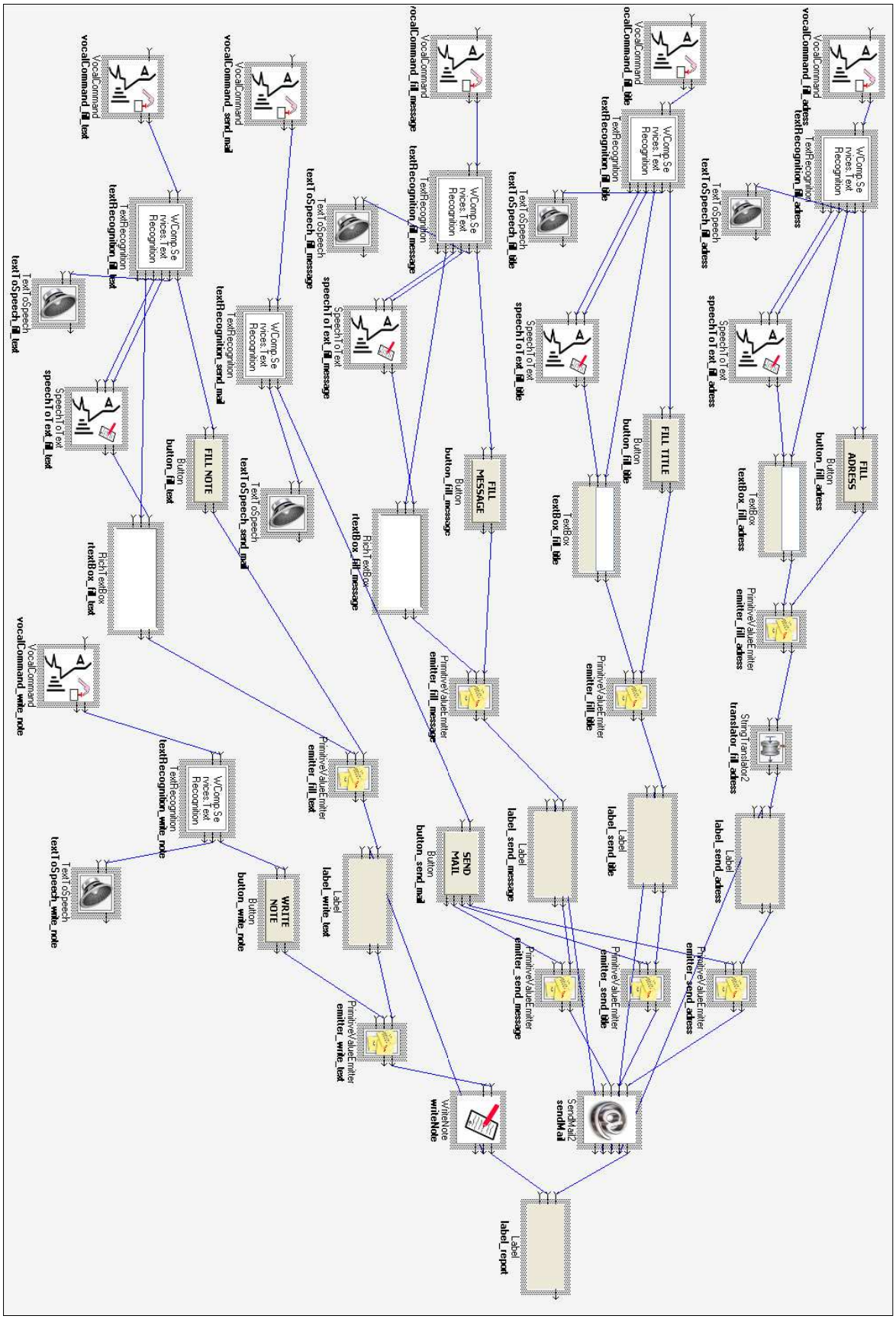
Annexe 11 : Code en C# du composant TextRecognition (première version)

```
using System;
using WComp.Beans;
using WComp.EventedBeans;

namespace WComp.Services
{
	[Bean(Category="Services")]

	public class TextRecognition : EventedDrawable
	{
		private enum selectState {not_selected,selected};
		private string wordkey;
		private selectState state = selectState.not_selected;

		public string keyword{
			get{	return wordkey;	}
			set{	wordkey = value;	}
		}

		public void CommandRecognition(string s)  {
			string word = "dot " + keyword + " dot";

			if ((s == word) && (state == selectState.not_selected))
			{
				state = selectState.selected;
				FireDisplayReport(keyword + " button selected");
			}
			else if ((s == word) && (state == selectState.selected))
			{
				FireEmitCommand();
				FireDisplayReport(keyword + " recorded");
				state = selectState.not_selected;
			}
		}

		public delegate void StringValueEventHandler(string str);
		public delegate void EventHandler();

		public event StringValueEventHandler DisplayReport;
		public event EventHandler EmitCommand;

		private void FireDisplayReport(string s){
			if (DisplayReport != null)		{
				DisplayReport(s);
			}
		}
		private void FireEmitCommand() {
			if (EmitCommand != null)		{
				EmitCommand();
			}
		}

	}
}
```

Annexe 12 : Code en C# du composant TextRecognition (version finale)

```
using System;
using WComp.Beans;
using WComp.EventedBeans;

namespace WComp.Services
{
	[Bean(Category="Services")]

	public class TextRecognition : EventedDrawable
	{
		private enum selectState {not_selected,selected};
		private string wordkey;
		private selectState state = selectState.not_selected;

		public string keyword{
			get{	return wordkey;	}
			set{	wordkey = value;	}
		}

		public void CommandRecognition(string s)  {
			string word = "dot " + keyword + " dot";

			if ((s == word) && (state == selectState.not_selected))
			{
				FireEmitText();
				state = selectState.selected;
				FireDisplayReport(keyword + " button selected");
				FireBool(true);
			}
			else if ((s == word) && (state == selectState.selected))
			{
				FireEmitCommand();
				state = selectState.not_selected;
				FireBool(false);
				FireDisplayReport(keyword + " recorded");
			}
			else if ((s == "dot erase dot") && (state == selectState.selected))
			{
				FireEmitErase();
			}
		}

		public delegate void StringValueEventHandler(string str);
		public delegate void BoolValueEventHandler(bool b);
		public delegate void EventHandler();

		public event StringValueEventHandler DisplayReport;
		public event BoolValueEventHandler EmitBool;
		public event EventHandler EmitCommand,EmitText,EmitErase;

		private void FireDisplayReport(string s){
			if (DisplayReport != null)		{
				DisplayReport(s);
			}
		}
		private void FireEmitCommand() {
			if (EmitCommand != null)		{
				EmitCommand();
			}
		}
		private void FireEmitText() {
			if (EmitText != null)	{
				EmitText();
			}
		}
		private void FireEmitErase() {
			if (EmitErase != null)	{
				EmitErase();
			}
		}
		private void FireBool(bool b) {
			if (EmitBool != null)	{
				EmitBool(b);
			}
		}
	}
}
```

Annexe 13 : Assemblage de l'application en mode vocal



BIBLIOGRAPHIE

[1] **M. WEISER**, "The Computer for the 21st Century", Scientific American, vol. 265, no. 3, pp. 94-104, 1991.

[2] **PUY Colin**, "L'informatique ambiante *"l'everyware "*". Master génie informatique. Grenoble : UFR Informatique & Mathématiques Appliquées de Grenoble, 17 p, 2006.

[3] **Canoe-techno&sciences**, "Microsoft dévoile l'ordinateur "Milan"" [en ligne]. Disponible sur <http://www2.canoe.com/techno/nouvelles/archives/2007/05/20070530-101156.html> (consulté le 06.06.2007).

[4] **CHEUNG-FOO-WOO D., LAVIROTTE S., RIVEILL M., TIGLI J.-Y.** , "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.

[5] **LAVIROTTE S., LINGRAND D., TIGLI J.-Y.** , "Définition du contexte : fonctions de coût et méthodes de sélection", Proceedings of the 2nd French-speaking conference on Mobility and ubiquity computing, pp. 9-12, 2005.

[6] **BAUDISCH P., TAN D., COLLOMB M., ROBBINS D., HINCKLEY K., AGRAWALA M., ZHAO S., RAMOS G.**, "*Phosphor : explaining transitions in the user interface using afterglow effects*", Proceedings of the 19th annual ACM symposium on User interface software and technology, pp. 169-178, 2006.

[7] **ERREY C.**, "*Visual and natural programming for non-programmers - a false hope?*" [en ligne]. Disponible sur <http://www.ptg-global.com/papers/development/visual-and-natural-programming.cfm> (consulté le 28.05.2007).

[8] **GARAVEL H., LANG F., RAYMOND P., SERWE W.**, "Méthodes formelles de développement". cours CNAM Grenoble 2007.

[9] **HOURDIN V., LAVIROTTE S., REY G., TIGLI J.-Y.**, "SLCA, une approche service pour l'informatique ambiante". Laboratoire I3S, équipe Rainbow, université de Nice-Sophia Antipolis/CNRS. Août 2008.

[10] **BUISSIERE N., CHEUNG-FOO-WOO D., HOURDIN V., LAVIROTTE S., RIVEILL M., TIGLI J.-Y.**, "Optimized Contextual Discovery of Web Services for Devices", IEEE Int. Workshop on Context Modeling and Management for Smart Environments, Oct 2007.

[11] **Universal Plug'n Play specification** [en ligne]. Disponible sur <http://www.upnp.org> (consulté le 15.11.2007).

[12] **CHEUNG-FOO-WOO D., BLAY-FORNARINO M., DERY A.-M., EMSELLEM D., RIVEILL M., TIGLI J.-Y.**, "Composition de contrats pour la Fiabilité d'ARchitecture Orientées Services : Identification des modalités de prise en charge des contrats pour chaque plate-forme cible". Projet FAROS, pp 27-45, Juin 2007.

[13] **Device Profile for Web Services specification**, [en ligne]. Disponible sur <http://specs.xmlsoap.org/ws/2006/02/devprof/devicesprofile.pdf> (consulté le 27.11.2007).

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

[15] **ROMAN M., HESS C., CERQUEIRA R., RANGANATHAN A., CAMPBELL R., NAHRSTEDT K.**, "Gaia : a middleware platform for active spaces", ACM SIGMOBILE Mobile Computing and Communications Review, vol. 6, no. 4, pp. 65-67, 2002.

[16] **YAU S., KARIM F., WANG Y., Wang B., GUPTA S.**, "*Reconfigurable Context-Sensitive Middleware for Pervasive Computing*", IEEE Pervasive Computing, vol. 1, no. 3, pp. 33-40, 2002.

[17] **BELLAVISTA P., CORRADI A., MONTANARI R., STEFANELLI C.**, "Context-Aware Middleware for Resource Management in the Wireless Internet", IEEE Transactions on software engineering, vol. 29, no. 12, pp. 1086-1099, 2003.

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

[19] **DURAN-LIMON H., BLAIR G., FRIDAY A., GRACE P., SAMARTZIDIS G., SIVAHARAN T., WU M.**, "*Context-aware middleware for pervasive and ad hoc environments*", *Computing Department, Lancaster University*, Bailrigg, Lancaster, UK, 2000.

[20] **SOUSA J., GARLAN D.**, "Aura : An Architectural Framework for User Mobility in Ubiquitous Computing Environments", Software Architecture : System Design, Development and Maintenance : IFIP 17th World Computer Congress-TC2 Stream/3rd Working IEEE/IFIP Conference on Software Architecture (WICSA3), August 25-30, 2002, Montréal, Québec, Canada, 2002.

[21] **SOUSA J., GARLAN D.** , "From Computers Everywhere to Tasks Anywhere : The Aura Approach", School of Computer Science, Carnegie Mellon University, [en ligne]. Disponible sur <http://www.cs.cmu.edu/aura> (consulté le 25/05/2008).

[22] **BELHANAFI N., TACONET C., BERNARD G.**, "CAMidO, A Context-Aware Middleware based on Ontology meta-model", Workshop on Context Awareness for Proactive Systems, pp. 93-103, 2005.

[23] **CAPRA L., EMMERICH W., MASCOLO C.**, "Re_ective Middleware Solutions for Context-Aware Applications", Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns, pp. 126-133, 2001.

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

[25] **SUN Microsystems JavaBeans 1.01 specification** [en ligne].1997. Disponible sur <http://java.sun.com/products/javabeans/> (consulté le 10/01/2008)

[26] **CHEUNG-FOO-WOO D., LAVIROTTE S., RIVEILL M., TIGLI J.-Y.**, "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.

[27] **CHEUNG-FOO-WOO D., BLAY-FORNARINO M., DERY A.-M., EMSELLEM D., RIVEILL M., TIGLI J.-Y.**, "Langage d'aspects pour la composition dynamique de composants embarqués", RSTI-Série L'Objet, vol.8, no 3, pp 89-112, 2002.

MEMOIRE D'INGENIEUR C.N.A.M. en INFORMATIQUE

La continuité de services en informatique ambiante : conception et adaptation d'applications sur WComp.

Eric Callegari **Grenoble, le 14 janvier 2009**

Résumé

La continuité de service est un thème essentiel du domaine des réseaux informatiques. En informatique ambiante, c'est à dire dans un monde où les dispositifs informatique sont de plus en plus nombreux et hétérogènes, elle nécessite une nouvelle approche. La notion de contexte d'une application entre en jeu et les applications doivent dorénavant s'y adapter. Divers paradigmes sont abordés dans le cadre de la conception d'applications et de leur adaptation au contexte, comme par exemple les architectures orientées service et la programmation orientée composants. De nombreuses recherches existent dans ce domaine, parmi lesquelles les travaux autour de la plateforme WComp. Cette dernière est particulièrement adaptée à la conception d'applications sous forme d'assemblages de composants et à l'adaptation au contexte grâce aux aspects d'assemblage, qui modifient la structure des assemblages. L'expérimentation de cette plateforme permet de se rendre compte de son potentiel et met en lumière les problèmes auxquels il faudra apporter des solutions pour rendre la plateforme WComp plus performante en termes d'adaptation des applications à leur contexte.

Mots-clés : continuité de service, informatique ambiante, contexte, adaptation au contexte, plateforme WComp, composant, assemblage, aspect d'assemblages.

Summary

Service continuity is an essential topic in the field of computer networks. In ubiquitous computing, say in a world where computer devices are more and more numerous and heterogeneous, it needs a new approach. The concept of context of an application comes into play and applications have to adapt to it. Many paradigms are addressed in the domain of conception of applications and their adaptation to the context, like for example, service oriented architectures and component oriented programming. Numerous researches exist in this field, among which are the works around WComp framework. This framework is particularly suitable for the conception of applications built like assemblies of components and the adaptation to the context thanks to aspects of assembly, which modify the structure of assemblies. Testing this framework allow to acknowledge its potential and to bring to light problems that will need to be solved in order to make WComp more efficient in terms of adaptation of applications to their context.

Keywords : service continuity, ubiquitous computing, context, adaptation to the context, WComp framework, component, assembly, aspect of assembly.