الجـمـهـوريـــة الجـــــزائـريـة الديمـقـراطيـة الـشـــــعبيـة

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

وزارة التـعـليـم العالي و البحث العـلـمـي

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

جـامعة أبي بكـــر بـلـقـايد– تـلمسـان –

Université Aboubakr Belkaïd– Tlemcen –
Faculté de TECHNOLOGIE

THESE

Présentée pour l'obtention du **grade** de **DOCTEUR 3ème cycle**

En : **Télécommunications**

Spécialité : **Télécommunications**

Par : **BAROUDI Mohammed Yassine**

Thème

Contribution au développement d'une architecture d'adaptation dynamique de services

Soutenue publiquement, le 09 / 11 / 2017 , devant le jury composé de :

BENMAMMAR Badr	**MCA**	**Univ. Tlemcen**	**Président**
BENDIMERAD Fethi Tarik	**Professeur**	**Univ. Tlemcen**	**Directeur de thèse**
BENAMAR Abdelkrim	**MCA**	**Univ. Tlemcen**	**Co-Directeur de thèse**
BENSLIMANE Sidi Mohammed	**Professeur**	**ESI de Sidi Bel Abbès**	**Examinateur**
BOUKLI HACENE Sofiane	**MCA**	**Univ. Sidi Bel Abbès**	**Examinateur**
MERIAH Sidi Mohamed	**Professeur**	**Univ. Tlemcen**	**Examinateur**

Résumé

Depuis plusieurs années les systèmes informatiques ont émergé, ils occupent désormais une place dans notre vie quotidienne. Les besoins grandissants et sans cesse croissants des utilisateurs des systèmes informatiques, ont engendré le développement d'applications de plus en plus grandes et complexes. La complexité de ces applications pose des problèmes tels que la réutilisation, l'installation, l'administration, l'évolution et l'adaptation dynamique.

Notre travail propose une architecture logicielle qui rend possible l'adaptation dynamique de services construits par assemblage de composants, en fonction de contextes d'utilisation variés.

Mots clés : Service, adaptation dynamique, composant.

Abstract

For several years computer systems have emerged, they now have a place in our daily lives. The growing needs and constantly growing users of computer systems have led the development of applications increasingly large and complex. The complexity of these applications poses problems such as reuse, installation, administration, development and dynamic adaptation.

Our work proposes a software architecture that enables the dynamic adaptation of services built by assembling components, depending on a variety of use environments.

Keywords: Service, dynamic adaptation component.

REMERCIEMENTS

J’adresse tout d’abord mes remerciements à Monsieur Fethi Tarik BENDIMERAD, Professeur des universités, directeur de thèse et ex directeur du Laboratoire de Télécommunication de Tlemcen (LTT), de m’avoir accueilli au sein de son laboratoire ainsi que pour avoir conduit et guidé mes travaux. Je remercie Dr Abdelkrim BENAMAR chef de département d’informatique à l’université Abou Bakr Belkaid, pour avoir encadré cette thèse, ces conseils précieux m’ont permis d’améliorer mes connaissances et d’aboutir à la production de ce travail. Je voudrais aussi le remercier pour le temps qu’il m’a accordé tout au long de ces années. Mes remerciements vont également à tous les collègues du laboratoire LTT pour les moments agréables partagés durant cette période. Je tiens à exprimer ma profonde gratitude et mes remerciements les plus sincères à Monsieur Cheikh Mohammed El Amine professeur des universités, Doyen de faculté de Technologie pour son aide diverses occasions et de leur encouragement continu. Je tiens aussi à remercier Philippe Dr Roose, pour m'avoir donné la chance d'effectuer un stage au sein de Laboratoire d'Informatique de l'UPPA, pour ses précieux conseils tant au niveau de scientifique qu’au niveau professionnel. Mes plus sincères remerciements vont également aux membres du jury, qui ont accepté d’évaluer mon travail de thèse. Je remercie Mr BENMAMMAR Badr, maître de conférences à l’université de Tlemcen, de m’avoir fait l’honneur de présider le jury de ma soutenance. Merci également à Mr MERIAH Sidi Mohamed, Professeur à l’université de Tlemcen, Mr BENSLIMANE Sidi Mohammed, professeur à l’ESI de Sidi Bel Abbès, Mr BOUKLI HACENE Sofiane, maître de conférence à l’université de Sidi Bel Abbès d’avoir accepté d’examiner ce travail et de faire partie de mon jury de thèse. À tout le jury, j’exprime ma reconnaissance pour les remarques, les questions et les recommandations qui ont été formulées. Je remercie aussi Dr Mostafa El Habib Dahou, et Dr Fayssal Beloufa qu’ils ont été les plus bénéfiques pour moi, merci pour leur disponibilité, leur écoute, leurs commentaires et leurs précieux conseils.

Un grand merci à tous mes amis Hakim, Yazid, Dris, Issam, Ahmed, Amine et tous ce qui m’ont aidé de loin ou de près.

Une grande reconnaissance à tous mes enseignants, du primaire à l’université, de m’avoir inculqué l’amour de la connaissance.

Enfin, mes remerciements vont à ma famille, mes parents, mon frère, mes sœurs, ma femme et mon fils pour avoir cru en moi et pour m'avoir soutenu tout au long de ces années.

Dédicaces

Je dédie cette thèse à mes parents, mon frère, mes sœurs, ma femme et mon fils pour avoir cru en moi et pour m'avoir soutenu tout au long de ces années.

A toute ma famille et tous mes amis

Il me serait difficile de vous citer tous, vous êtes dans mon cœur, affectueusement.

A tous ceux qui, par un mot, m’ont donné la force de continuer …..

Table de matières

Liste des Figures

ADL : Architecture Description Language
API : Application Programming Interface
BPEL : Business Process Execution Language
CCM : Corba Component Model
CIDL : Component Implementation Definition Language
CIF : Component Implementation Framework
CORBA : Common Object Request Broker Architecture
DME : Description Modification Elements
EJB : Entreprise Java Beans
EMF : Eclipse Modeling Language
GPRS : Global Pocket Radio Service
GSM : Global System for Mobile communications
IBM : International Business Machines
IDL : Interface Definition Language
IDM : Ingénierie Dirigée par les Modèles
IHM : Interface Homme-Machine
ISL : Interaction Specification Language
ISL4Wcomp : Interaction Specification Language for Wcomp
JAR : Java Archive
JVM : Java Virtual Machine
LDAP : Lightweight Directory Access Protocol
LOO : Langage Orienté Objet
MDA : Model Driven Architecture
MOF : Meta Object Facility
MVC : Multi Visionning Connector
OMG : Object Management Group
ORB : Object Request Broker
OSGI : Open Service Gateway Initiative
PIM : Platform Independent Model
POA : Portable Object Adapter
POO : Programmation Orientée Objet
PSM : Platform Specific Model
QdS : Qualité de Service
RMI : Remote Method Invocation
SCA : Service Component Architecture
SME : Specification Modification Elements
SOA : Service Oriented Architecture
SOAP : Simple object Access Protocol
UDDI : Universal Description Language
UML : Unified Modeling Language
URL : Uniform Resource Locator
WS-BPEL : Web Services Business Execution Language

WS-CDL : Web Services Choreography Description Language
WSDL :Web Service Description Language
WSFL : Web Services Flow Language
XML : eXtensible Markup Language

Introduction Générale

Contexte général

Dans les dernières années, l'informatique a considérablement évolué. L'évolution technologique réalisée dans le domaine de l'informatique ne cesse de croître et marque profondément notre société actuelle. Les systèmes logiciels sont devenus indispensables dans de nombreux domaines d'application, les besoins grandissants et sans cesse croissants des utilisateurs, ont engendré le développement d'applications de plus en plus complexes. La construction d'applications modernes présente donc des défis de conception, de développement, d'exécution et de maintenance, imposant de redéfinir comment une application est conçue, développée, exécutée et gérée à l'exécution.

- **L'approche à composant et l'approche à service**

De nombreux travaux de recherche s'intéressent à la construction d'applications logicielles afin de proposer des nouvelles approches qui répondent aux besoins et aux défis des applications. L'approche à composants (Component Based Software Engineering - CBSE) (Szyperski et al, 2002), apparue autour du milieu des années 90, a été motivée d'un côté par des arguments économiques comme la réduction du temps et des coûts de développement des applications, et d'un autre côté pour éliminer les limitations de l'approche à objets (Taylor et al, 1998). Elle promet la construction d'applications par réutilisation et par assemblage (composition) de composants. De façon simpliste, un composant peut être décrit comme une brique logicielle composable avec d'autres composants et réutilisable dans la construction de différentes applications. Pour permettre son utilisation, un composant précise par des interfaces, les fonctionnalités qu'il fournit mais aussi ses dépendances fonctionnelles. L'approche à composants propose une séparation claire entre les interfaces fournies et le code réalisant l'implémentation. L'approche à composants est centrée implémentation, abordant principalement le développement et la composition structurelle de composants. Une application est ainsi composée structurellement à travers les interfaces des composants : les implémentations de composants constituant l'application sont liées entre elles d'une interface requise à une interface fournie. En conséquence, différentes compositions d'une application (nommées configurations) peuvent être possibles en

raison des différentes implémentations de composants disponibles au moment de la composition.

L'approche à composants a connu une grande popularité et des nombreux modèles de composants ont été proposés tant dans le monde académique qu'industriel (Kung-Kiu, 2006). Cependant, ces modèles ne traitent pas en général les besoins de flexibilité et de dynamisme des applications modernes : les applications sont généralement composées au développement ou au déploiement ; une fois une application composée, les changements dynamiques dans son architecture ne sont pas possibles ou sont difficiles à réaliser, soit parce que les modèles ne supportent pas la description d'architectures dynamiques, ou parce que les technologies sous-jacentes d'exécution ne supportent pas la réalisation de changements dynamiques. Les architectures des applications sont donc relativement statiques.

L'approche à services (Service Oriented Computing - SOC) (Papazoglou, 2003) est apparue plus récemment pour répondre aux besoins de flexibilité, de dynamisme, d'hétérogénéité et de distribution des applications. Cette approche propose de construire des applications à partir d'éléments logiciels, nommés services, qui peuvent être fournis par des tiers et qui peuvent évoluer de façon dynamique. L'approche à services, comme l'approche à composants, propose une séparation claire entre la description des fonctionnalités fournies et l'implémentation d'un service. A partir d'une description, un client peut rechercher, découvrir, sélectionner et invoquer un service. L'approche à services met l'accent sur le fait que ce cadre d'interaction – recherche, découverte, sélection, invocation – peut être effectué à l'exécution afin de pouvoir utiliser les services disponibles à un moment précis. L'approche à services est centrée instance, abordant principalement l'exécution de services.

L'approche à services fournit ainsi deux propriétés importantes : le faible couplage entre clients et fournisseurs de services, et la liaison tardive. Grâce à ces propriétés, l'approche à services permet la construction d'applications flexibles et dynamiques par composition de services.

Les principes de l'approche à services ont été mis en œuvre par diverses approches académiques et industrielles. Certaines approches proposent des modèles pour le développement de services et/ou pour sa composition. Quelques-unes proposent des langages pour la description de l'architecture des applications. D'autres offrent des

plates-formes pour supporter leur exécution. Cependant, en général, les approches permettant la composition ne permettent pas le dynamisme : la composition est relativement statique empêchant toute substitution de composants à l'exécution. Inversement, les approches proposant des mécanismes de gestion du dynamisme ne proposent pas de mécanismes de composition d'applications et les applications sont donc difficiles à administrer. Il existe en effet une dissociation entre le développement et l'exécution des applications : les informations présentes lors du développement d'une application, notamment les concepts d'implémentation, de dépendance, d'architecture, ne sont pas connus à l'exécution. Cette séparation de la connaissance entre le développement et l'exécution rend difficile de contrôler et de garantir l'exécution des applications. La construction et la maintenance d'applications à services reste ainsi une tâche assez complexe. Les besoins sans cesse croissants des utilisateurs, et les changements rapides des environnements, amènent les développeurs à concevoir et implémenter des applications pouvant s'adapter dynamiquement à ces besoins et contraintes.

L'adaptation est une opération qui consiste à apporter des modifications à une application ou à un système informatique apte à assurer ses performances dans des conditions particulières ou nouvelles pour un contexte précis d'utilisation. Par dynamique, nous signifions la possibilité d'introduire des modifications dans une application en exécution sans l'arrêter.

- **L'adaptation dynamique**

L'adaptation consiste à rendre un système apte à assurer ses fonctions dans des conditions particulières ou nouvelles. Il existe plusieurs moments pour adapter une application. L'adaptation réalisée plus tardivement (par exemple à l'exécution) est plus puissante (par rapport à une adaptation réalisée au développement par exemple), mais aussi plus complexe à mettre en place.

L'adaptation dynamique permet de modifier la structure de l'application et de faire évoluer les algorithmes qu'implantent les composants pendantl'exécution et sans avoir à tout redémarrer. L'adaptation dynamique donne donc plus de garanties pour satisfaire les exigences envers l'application de manière continue.

Différents types d'adaptation ont été mentionnés dans la littérature. Certaines adaptations concernent les clients de l'application et sont généralement difficiles à réaliser, d'autres affectent l'architecture de conception ou l'architecture de déploiement.
En effet, chaque système logiciel se décompose en objets, composants, services selon son architecture et la technologie utilisée pour sa mise en œuvre. Chacune de ces entités peut être le sujet d'adaptation (attributs, méthodes, composition…).
Nous distinguons plusieurs types d'adaptations possibles de l'application dans les travaux existants (Ketfi et al, 2002). Généralement, l'adaptation concerne les paramètres de configuration, l'implémentation, les interfaces, la géométrie et la structure de l'application.

- Adaptation de paramètres (Zhang et al, 2003), (Haifeng et al, 2002) : Ce type d'adaptation consiste à modifier les valeurs de variables qui influent sur le comportement de l'application sans changer les algorithmes.
- Adaptation d'implémentation (Capra et al, 2003), (Chen et al, 2001), (Efstratiou et al, 2002) : Cette adaptation permet de modifier l'implémentation du composant (le code du composant) sans changer ni les interfaces ni les connexions avec les autres composants de l'application.
- Adaptation d'interfaces (Heineman, 2000) : Chaque composant de l'application fournit des services à travers des interfaces bien définies. L'adaptation d'interfaces consiste à modifier les services fournis par un composant en changeant ses interfaces.
- Adaptation de géométrie (Tilevich et al, 2009) : L'adaptation de géométrie modifie la distribution géographique d'une application. En effet, ce type d'adaptation impacte uniquement l'emplacement des composants qui peuvent changer de localisation en migrant d'une machine vers une autre. Cette adaptation est appelée aussi adaptation de localisation ou encore la migration.
- Adaptation de structure (Cassagnes et al, 2009), (Dwling et al, 2004), (Garlan et al, 2004), (Layaida et al, 2005) : Ce type d'adaptation modifie la topologie de l'application. Le changement porte sur la structure de l'application en termes de composants et de connexions. Les opérations de base pour réaliser les modifications sont : l'ajout d'un composant, la suppression d'un composant, l'ajout d'une connexion et la suppression d'une connexion.

Problématique et solution proposée

L'objectif de ce document est de proposer une architecture logicielle permettant l'adaptation dynamique de services construits par assemblage de composants, en fonction de contextes d'utilisations variés. Le contexte d'utilisation concerne les besoins de l'utilisateur.

L'approche à service propose de construire des applications à partir d'entités logicielles qui peuvent être fournies par des organisations tierces et qui peuvent évoluer dynamiquement. Ces entités sont nommées services. Un service fournit une fonctionnalité qui, pour être mise à disposition, est spécifiée, de façon syntaxique et/ou sémantique, par une description de services. La description de services permet aux éventuels clients de rechercher, découvrir et par la suite appeler un service. De manière idéale, ce protocole (découverte, sélection, invocation) peut être réalisé à l'exécution, cela donnant lieu à la création d'applications dynamiques utilisant des services disponibles à un moment précis, et donc donnant une grande flexibilité à l'application.

Nous supposons qu'une application de service est construite par assemblage de composants, nous considérons que l'adaptation de celle-ci s'effectue au niveau de son architecture par ajout/retrait/remplacement de ses composants.

Notre proposition est basée sur une méta-description de l'ensemble service-contexte dans lequel nous avons un ensemble limité des axiomes d'adaptation basés sur la sémantique du service. Pour prouver le fonctionnement de cette architecture nous avons implémenté un prototype de service de forum. Ce service est crée initialement pour des utilisateurs francophones mais l'architecture proposée permet d'adapter ce service par ajout dynamique d'un composant de traduction si la langue de l'utilisateur n'est pas le français.

Organisation du manuscrit

En plus de l'introduction, ce document est composé de quatre chapitres : Deux chapitres introduisent les grandes lignes dans lesquelles se place cette thèse et dressent un état de l'art des approches à composants et à services d'une part, et l'adaptation dynamique

d'autre part. Les deux chapitres suivants présentent notre approche de contribution au développement d'une architecture d'adaptation dynamique de services.

En ce qui concerne ces chapitres de manière plus détaillée :

Chapitre1 : présente les architectures et approches de développement des systèmes informatiques de grandes tailles et de plus en plus complexes, à savoir l'approche orientée composant, orientée service, et l'approche à composant à service. De plus il définit pour chacune des approches ses avantages et ses limites. Ce chapitre étudie aussi quelques modèles utilisant les approches à service et à composant.

Chapitre2 : décrit la notion d'adaptation dynamique de manière générale, les raisons de l'adaptation, les types d'adaptation, ainsi que les approches à service dynamique.

Chapitre3 : se focalise sur la composition des services web, il définit la notion de composition, les approches et les différents types de composition de service, ainsi que les langages de composition et les défis reliés à la composition de services. Il présente aussi les concepts, motivations, et mécanismes d'adaptation de services.

Chapitre4 : définit la notion de contexte, présente les différentes catégories contextuelles, ainsi que les systèmes sensibles au contexte, il présente aussi une architecture permettant l'adaptation dynamique de service. Cette proposition est basée sur une méta-description de l'ensemble service-contexte dans lequel nous avons un ensemble limité des axiomes d'adaptation basés sur la sémantique du service. Pour prouver le fonctionnement de cette architecture nous avons implémenté un prototype de service de forum.

Enfin, le document se termine par une synthèse et présente les perspectives pour ce travail.

Chapitre 1 :
Approche à composant et approche à service

1 Introduction

Depuis plusieurs années les systèmes informatiques ont émergé, ils occupent désormais une place dans notre vie quotidienne. Les besoins grandissants et sans cesse croissants des utilisateurs des systèmes informatiques, ont engendré le développement d'applications de plus en plus complexes.

La complexité de ces applications pose des problèmes tels que la réutilisation, l'installation, l'administration et l'évolution des applications. Pour pallier ces problèmes, plusieurs approches ont vu le jour.

L'approche à composant est basée sur des techniques et des langages permettant de construire et de structurer un système comme un assemblage de briques logicielles, dont les dépendances sont clairement identifiées. Ces briques logicielles sont appelées « composants ». Le système résultant de l'approche à composants possède les propriétés de réutilisabilité (réutilisation de composants), de partitionnement (découpage en composants), et d'adaptation (configuration des composants) (Tournier et al, 2005), en plus de la possibilité de remplacer un composant par un autre.

L'approche à service quant à elle, propose de construire des applications à partir d'entités logicielles qui peuvent être fournies par des organisations tierces et qui peuvent évoluer dynamiquement. Ces entités sont nommées services. Un service fournit une fonctionnalité qui, pour être mise à disposition, est spécifiée, de façon syntaxique et/ou sémantique, par une description de services. La description de services permet aux éventuels clients de rechercher, découvrir et par la suite appeler un service. De manière idéale, ce protocole (découverte, sélection, invocation) peut être réalisé à l'exécution, cela donnant lieu à la création d'applications dynamiques utilisant des services disponibles à un moment précis, et donc donnant une grande flexibilité à l'application.

Dans ce chapitre, nous présentons un état de l'art sur les approches à composants et à services passant par les approches : procédurale et orientée objet.

2 Approche procédurale

Les premiers langages de programmation étaient généralement constitués d'une suite d'instructions s'exécutant de façon linéaire.

Dans ce contexte, le lancement d'un programme débutait par l'exécution de la première instruction du fichier source et se poursuivait ligne après ligne jusqu'à la dernière instruction du programme.

Cette approche linéaire, bien que simple à mettre en œuvre, a très rapidement montré ses limites. En effet, les programmes monolithiques (construits d'un seul bloque) de ce type ne se prêtent guère à l'écriture de grosses applications et ne favorisent absolument pas la réutilisation du code. En conséquence sont apparus d'autres langages (tels que le C) qui proposaient une approche radicalement différente : l'approche procédurale.

L'approche procédurale (ou modulaire) consiste à découper un programme en une série de fonctions (ou procédures). Ces fonctions contiennent un certain nombre d'instructions qui ont pour but de réaliser un traitement particulier. Le calcul de la circonférence d'un cercle, l'impression de la fiche de paie d'un salarié sont des exemples de traitements qui peuvent être symbolisés par des fonctions. Dans le cas de l'approche procédurale, un programme correspond à l'assemblage de plusieurs fonctions qui s'appellent entre elles.

L'approche procédurale présente également des inconvénients.

D'une part, elle a tendance à dissocier les données des fonctions qui les manipulent. D'autre part, elle ne fournit pas de technique permettant de favoriser la spécialisation des fonctions existantes.

3 Approche orientée objet

Dès le début des années 80, une nouvelle technologie a été créée afin d'améliorer ce mode de fonctionnement : la *programmation orientée objet* (POO). Cette technique est destinée à améliorer la réutilisation des outils existants et à favoriser la spécialisation de ces mêmes outils. Dans ce contexte, une nouvelle entité, appelée *objet*, a été créée afin de regrouper les fonctions et les données. Comme vous le découvrirez dans ce qui suit, ces objets sont devenus des acteurs incontournables des développements informatiques.

Le terme objet est parfois abusivement employé pour désigner ce qu'on appelle en C++ une classe. La classe représente le moule, le gabarit qui permet de créer les objets. En d'autres termes, la classe correspond dans les faits à la déclaration d'un type de donnée qui permet de créer des variables, les objets (appelés également instances de classe). Une classe est également appelée famille d'objets. Par exemple, une classe permettant

de gérer des voitures regrouperait les caractéristiques d'une voiture et permettrait de créer plusieurs objets de ce type. Les variables qui appartiennent à une classe sont appelées des données membres.
Les fonctions qui appartiennent aux classes sont appelées des fonctions membres ou des méthodes.
Un programme qui utilise l'approche objet s'appuie sur trois techniques fondamentales :

3.1 L'encapsulation

Elle permet de réunir au sein d'une même entité appelée classe des données (*variables*) et des traitements (*fonctions*).Cette technique s'accompagne également d'un système de protection qui permet de contrôler l'accès aux données et traitements de la classe. Cela signifie qu'il est possible de cacher le fonctionnement interne d'une classe et ainsi éviter que les objets de cette classe soient utilisés de façon non conforme.

3.2 L'héritage

Il permet de bénéficier du patrimoine existant, c'est-à dire de réutiliser les classes existantes. Dans ce contexte, il est possible de définir une nouvelle classe à partir d'une autre. Cela signifie que la classe descendante (ou dérivée) pourra bénéficier des attributs (variables) et des comportements (fonctions) de la classe de base dont elle hérite (ou dérive). Vous découvrirez également que les classes dérivées peuvent contenir d'autres éléments permettant de spécialiser la classe de base. Grâce à cette technique, il est donc possible de définir des arborescences (ou hiérarchie) de classes qui regroupent des classes de plus en plus spécialisées.

3.3 Le polymorphisme

Il s'applique aux fonctions membres des classes et permet à deux objets (appartenant à deux classes différentes) de réagir différemment au même appel de méthode. Cette technique s'utilise conjointement avec l'héritage et représente sans conteste le plus grand apport des langages orientés objet (LOO).

4 Approche orientée composant

La réutilisation a été toujours un défi majeur depuis le début du génie logiciel (Dieng, 2010). C'est pourquoi, de nombreuses approches (Taylor, 1998), (Booch, 1993), (Szyperski et al, 2002), (Heineman et al, 2001) ont été proposées pour apporter des solutions à cette problématique de réutilisation. Ainsi, l'approche à composant a été définie pour améliorer la réutilisation du code des applications logicielles. Pour ce faire, cette approche promeut la construction d'une application à partir d'un ensemble de briques logicielles bien définies et indépendantes, appelées composants (Szyperski et al, 2002), (Heineman et al, 2001).

4.1 Définition de composant

Il n'existe pas une définition standardisée de la notion de composant, nous adoptons la définition donnée par *Szyperski*, souvent citée dans la littérature (Macedo, 2004), (Tournier, 2005), (Cervantes, 2004), (Ketfi, 2004).

"A Software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A Software component can be deployed independently and is subject to composition by third parties".

Un composant logiciel est une unité de composition ayant des interfaces spécifiées de façon contractuelle qui possède uniquement des dépendances de contexte explicites. Un composant peut être déployé de manière indépendante et est sujet à composition par des tierces (Cervantes, 2004).

L'intérêt de cette définition est qu'elle contient plusieurs concepts importants (interfaces, contrat, composition, déploiement, …) qui caractérisent un composant et qui seront discutés dans ce chapitre.

De manière plus générale on peut dire (Projet accord, 2002) :

• un composant est un logiciel qui rend un service au moyen d'une ou de plusieurs interfaces externes.

• Un composant a vocation à être intégré dans plusieurs applications. Pour cela, il doit être suffisamment autonome et comporter toutes les informations lui permettant d'être assemblé par un tiers, éventuellement avec d'autres composants.

• Un composant réutilisable est un code compilé ou binaire, et doit renfermer toutes les informations nécessaires à son déploiement dans un environnement donné.

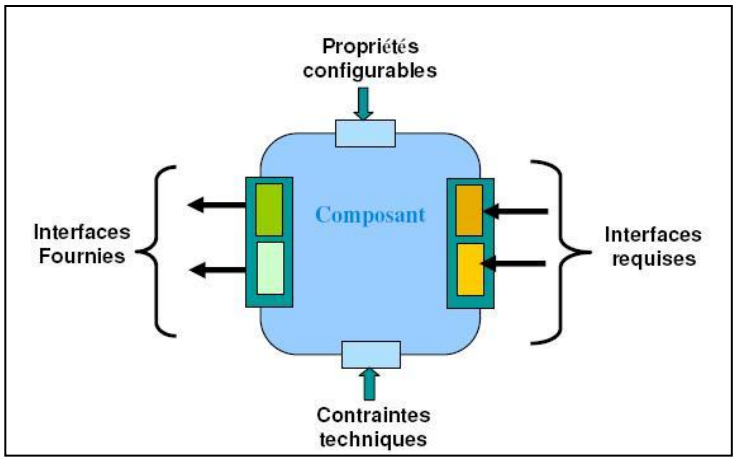

Figure 1: Architecture d'un composant logiciel

Comme le montre la figure 1, un composant logiciel possède, principalement, les trois éléments suivants (Hadjab, 2003) :

• interfaces : dans un composant on trouve deux types d'interface, les interfaces fournies, et les interfaces requises. Les interfaces fournies permettent d'accéder aux services fournis par le composant, tandis que les interfaces requises modélisent les dépendances externes du composant (Crnkovic et al, 2002).

• Les propriétés configurables : généralement, ce sont des attributs qui permettent d'adapter et de personnaliser le composant dans des contextes d'exécution spécifiques.

• Les contraintes techniques (Services) peuvent être : la sécurité, la persistance, les transactions, etc.

4.2 Concepts de base des composants

4.2.1 Type de composant

Un type de composant est caractérisé par trois éléments : ses interfaces, ses modes de coopération et ses propriétés configurables (Crnkovic et al, 2002):

A/ Interfaces

Les services offerts par un composant sont définis au travers de plusieurs interfaces (Projet accord, 2002). (Crnkovic et al, 2002) définit une interface de composant comme une spécification de ses points d'accès. Les clients accèdent aux services fournis par le composant en utilisant ces points. Une interface nomme seulement une collection d'opérations et fournit la description et les protocoles d'utilisation de ces opérations, elle ne fournit aucune implémentation.

Ces interfaces sont associées aux différentes facettes de l'activité du composant (interface métier, interface de configuration, de maintenance, ...). Par ailleurs, un composant s'appuie sur d'autres composants pour fonctionner. À cette fin, un composant définit des interfaces requises (les interfaces dont il a besoin dans le cadre de son exécution). Ces interfaces doivent être connectées à des interfaces offertes par d'autres composants.

La séparation entre implémentation et interfaces dans un composant permet de remplacer la partie implémentation sans modification des interfaces et vice versa, ce qui permet d'augmenter les performances de système sans reconstruire l'application et donner une certaine souplesse en termes d'adaptabilité (Crnkovic et al, 2002).

Les informations contenues dans les interfaces d'un composant facilitent la vérification de l'interopérabilité entre composants, et permettent à certaines propriétés d'être vérifiées plus tôt dans le cycle de conception (par exemple la vérification statique des erreurs de typage) (Macedo, 2004).

B/ Mode de coopération

Pour chacune des interfaces d'un composant, il est nécessaire de spécifier le mode de coopération. Il existe trois modes de coopération : le mode synchrone (par exemple l'invocation de méthode), le mode asynchrone (par exemple l'envoi d'un message) et le mode diffusion en continu (par exemple la communication par flots de données).

C/ Propriétés configurables

Ces propriétés permettent d'adapter une instance de composant en configurant son comportement. Le code du composant n'est pas modifié en fonction du besoin, mais paramétré, augmentant ainsi la réutilisabilité du composant.

4.2.2 Spécification et contrat

Une spécification de composant nomme toutes les interfaces auxquelles pourraient adhérer un composant et ajoute toutes les propriétés spécifiques au composant. Cette spécification décrit les services offerts par le composant, mais aussi ceux dont il a besoin pour fonctionner. Les spécifications représentent donc l'ensemble des contraintes que le composant impose à ses utilisateurs, ainsi que celles venant de son environnement qu'il s'impose de respecter (Projet Accord, 2002).

Le terme de contrat est de plus en plus utilisé pour décrire les propriétés nécessaires au bon fonctionnement du composant. Chaque composant est ainsi spécifié par deux types de contrat (Projet Accord, 2002):

• un unique contrat d'utilisation qui précise les services qu'il offre et qu'il requiert ;

• des contrats d'implantation qui décrivent les contraintes et les propriétés requises de l'environnement pour le déploiement.

4.2.3 Notion de port

Les interfaces peuvent être regroupées au sein de ports. Ces ports permettent de regrouper les interfaces relatives à une préoccupation donnée (par exemple, les interfaces d'administration, configuration, etc.). Un port se comporte comme un point d'interaction du composant avec l'environnement, les ports peuvent posséder à la fois des interfaces requises et fournies (Tournier, 2005).

4.2.4 Assemblage de composants

Un composant logiciel est une unité de réutilisation et d'intégration. Il est donc conçu pour fonctionner en coopération avec d'autres composants. Pour cela, il faut réaliser l'étape de composition ou d'assemblage d'un ensemble de composants caractérisés essentiellement par leurs interfaces offertes et leurs interfaces requises.

Un assemblage est donc un ensemble de composants qui coopèrent pour la construction d'une application logicielle (Crnkovec et al, 2002). Les liaisons entre composants sont les liaisons entre les services requis par les uns et les services offerts par les autres.

4.2.5 Notion de connecteur

La notion de connecteur recouvre l'ensemble des moyens nécessaires pour assurer l'interaction entre les composants par la connexion des interfaces requises et offertes (Projet Accord, 2002). Un connecteur relie une interface requise d'un composant à une interface fournie d'un autre composant (Tournier, 2005).

Les connecteurs sont proposés principalement pour faciliter l'assemblage des composants en jouant le rôle d'intermédiaires entre les composants non compatibles.

Les connecteurs sont des entités architecturales de communication qui modélisent de manière explicite les interactions (transfert de contrôle et de données) entre les composants. Ils contiennent des informations concernant les règles d'interactions entre les composants. Ils s'occupent de gérer les interactions (communication /coordination) entre les composants (Hadjab, 2003).

Les connecteurs sont classifiés (Mehta et al, 2000) selon les services qu'ils offrent, donc on trouve les services de communication, de coordination, de conversion et de facilitation. Par ailleurs d'autres services peuvent être offerts par les connecteurs, par exemple, la notion de MVC (Multi-Versioning Connector) (Rakic et al, 2001), qui permet l'utilisation simultanée de plusieurs versions d'un même composant, ce connecteur redirige les appels aux services vers la version du composant la plus adéquate. Les connecteurs peuvent aussi être utilisés comme moyen de détection et de réparation (self-healing mechanism) des anomalies au niveau des objets et des interactions des tâches dans les composants (Shin et al, 2005).

4.2.6 Architecture logicielle et langage de description d'architecture

Dans l'approche "composant", une application, construite par une collection de composants logiciels interconnectés à l'aide de connecteurs, est appelée une architecture logicielle (figure 2). En effet, l'architecture fournit la description de haut niveau de la structure d'un système ce qui permet de réduire sa complexité (El Boussaidi et al, 2006).

Pour mettre en évidence la structure d'une application, une architecture peut être décrite à l'aide de langages de description d'architecture ADL (Architecture Description Language). Les ADLs focalisent sur la structure de l'application globale à un haut

niveau d'abstraction plutôt que sur les détails d'implémentation (Garlan, 2000). Pour cela ils fournissent une syntaxe concrète et un cadre conceptuel pour la modélisation d'une architecture conceptuelle du système logiciel. Celle-ci repose généralement sur les modèles de base suivants :

• Les composants représentant des unités de calcul ou de stockage

• les connecteurs qui sont des composants particuliers employés pour modéliser les interactions entre les composants ainsi que les règles qui régissent ces interactions ;

• les configurations architecturales La configuration permet de décrire la structure globale d'un logiciel, c'est-à-dire l'assemblage des composants de ce logiciel sous forme de graphes de connexion formés de composants et de connecteurs.

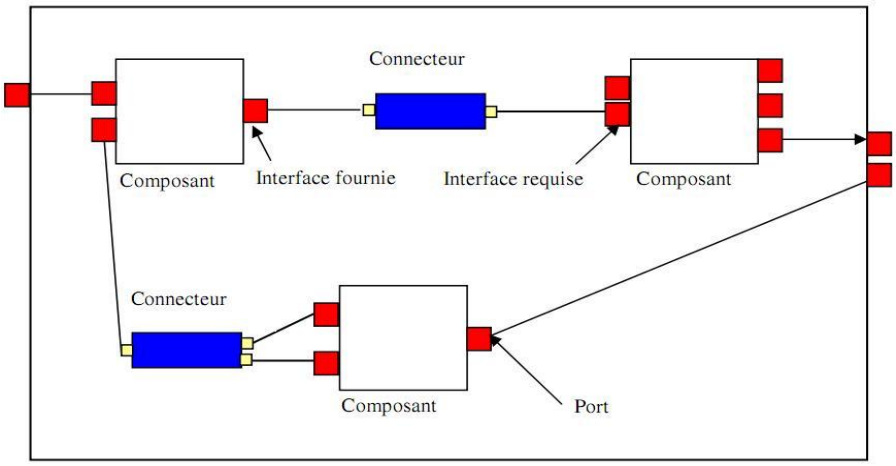

Figure 2: Architecture d'une application à base de composants

Les ADLs plus connus sont : Wright (Allen, 1997), ACME (Garlan et al, 2000), Rapide (Luckham et al, 1995), UniCon ((Shaw et al, 1995), et Darwin (Magee et al, 1995).

4.2.7 Modèle de composants et environnement d'exécution

Un modèle à composants définit la structure des composants et de leurs assemblages et permet de réaliser le développement suivant le cycle de développement propre à cette approche (Nano, 2004), c'est une spécification d'une infrastructure de support aux composants, telles que EJB, CCM, Fractal, .Net.

Un modèle à composants est accompagné par un environnement d'exécution qui fournit du support aux applications pendant l'exécution. L'environnement d'exécution est parfois comparé à un mini système d'exploitation car il est chargé de gérer des aspects

divers tels que le cycle de vie des instances des composants ou bien les propriétés non fonctionnelles (Cervantes, 2004).

4.3 Cycle de développement d'une application à base de composants

Le cycle de développement d'une application à base de composants passe par trois étapes principales (figure 3) (Cervantes, 2004), (Luckham et al, 1995) :

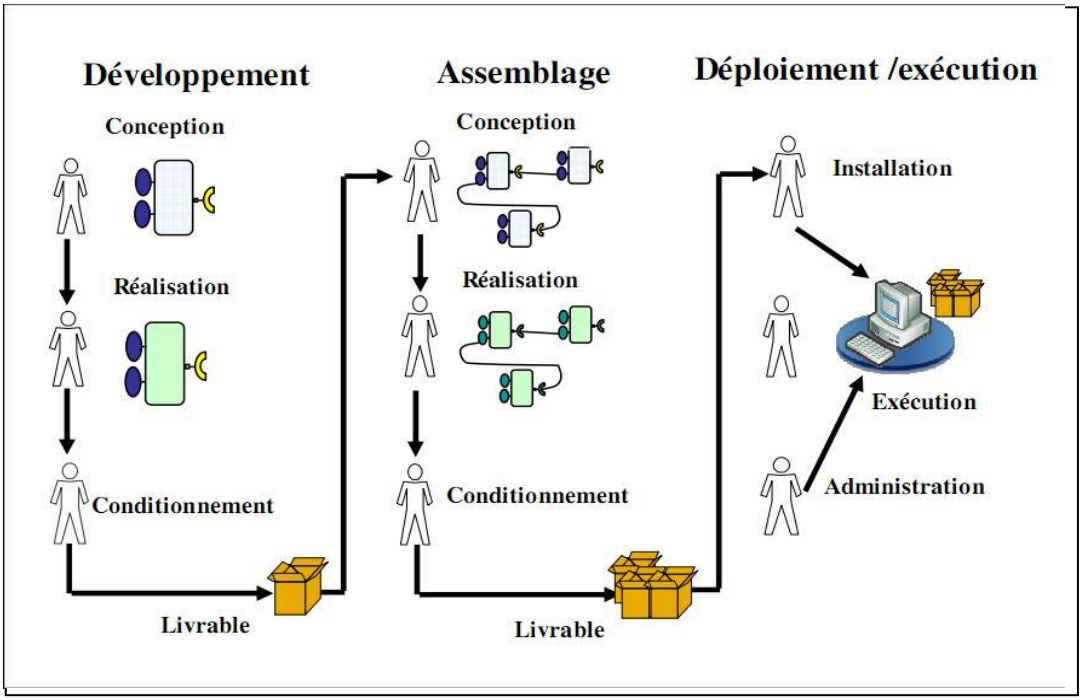

Figure 3: Cycle de vie de développement simplifié (Cervantes, 2004).

4.3.1 Développement des composants

Le développement de composants inclut les activités de *conception*, *réalisation* et *conditionnement* des composants destinés à être assemblés dans différentes applications. La conception inclut la spécification de la *vue externe* (interfaces et propriété de configuration) d'un composant ainsi que de son comportement. La réalisation, c'est l'implémentation effective des interfaces du composant (les fonctionnalités), un composant peut avoir plusieurs implémentations pour une même vue externe. Le conditionnement consiste à introduire le composant dans un *package* livrable, ce package permet le dépoilement du composant dans une application d'une manière indépendante (autonomie).

4.3.2 L'assemblage (composition)

L'étape d'assemblage suppose l'existence de composants développés auparavant qui sont utilisés par les acteurs de cette étape. L'assemblage (ou composition) de composants inclut les activités de *conception, réalisation* et de *conditionnement*. La conception d'un assemblage est réalisée en étudiant la manière de composer un ensemble de composants préexistants pour créer une composition représentant soit une partie ou bien la totalité d'une application, c'est-à-dire une architecture. La réalisation d'un assemblage consiste à l'écriture du code permettant de réaliser la composition des instances de composants. Lors du conditionnement, un assemblage est introduit dans un *paquetage d'assemblage* qui peut inclure les composants employés dans l'assemblage ainsi que des ressources propres à l'assemblage telles que des fichiers de configuration ou des fichiers binaires (librairies, images, sons, etc.).

4.3.3 Déploiement, exécution et administration

le déploiement c'est l'installation des paquetages contenant des compositions et des composants, dans cette étape des activités de configuration peuvent être réalisées. L'exécution est la mise en marche de l'application à ce moment là, l'environnement d'exécution est actif. L'administration inclut l'application de mises à jour, d'adaptation et de maintenance, ainsi que la désinstallation d'une application et de ses composants.

4.4 Etude de quelques modèles à composants

Plusieurs modèles de composants on été définies, des standards industriels tel que EJB de Sun MicroSystems ,CCM de l'OMG et .NET de Microsoft (Baroudi et al, 2013) d'autre en voix de standardisation tel que Fractal (Bruneton et al, 2004). Nous consacrons cette partie, pour présenter une brève description des architectures de ces modèles.

4.4.1 Le modèle Entreprise java Beans (EJB)

EJB est un modèle à composant industriel, dont la première spécification est apparue en 1998 par Sun Microsystems, propose un modèle de composants pour la construction

d'applications réparties côté serveur dans le cadre des architectures n-tiers. Une telle application est constituée de composants appelés Beans implémentés en Java.
Un *entreprise bean* est un *composant côté serveur* qui encapsule la logique métier d'une application. La logique métier est le code qui accomplit le but de l'application.

4.4.2 le modèle CORBA Component Model (CCM)

CCM est un modèle à composant industriel, spécifié par l'Object Management Group (OMG) en 2002, ce modèle est basé sur CORBA qui permet l'utilisation d'environnements d'exécution hétérogènes (Allen, 1997). Le CCM se décompose en deux niveaux : un niveau basique, qui permet essentiellement de « componentiser » (rendre des composants) des objets CORBA, et un niveau étendu, beaucoup plus riche et intéressant. Tout comme les EJB de Sun, auxquels correspond la version basique du CCM en Java, la technologie CCM repose sur l'utilisation de conteneurs pour héberger les instances de composants et faciliter leur déploiement (Projet Accord, 2002).
CCM propose toute une structure pour définir un composant, son comportement, son intégration, et son déploiement dans l'environnement distribué CORBA (Macedo, 2004).
CCM définit les types de composants, ainsi que leurs implantations, à l'aide de langages déclaratifs. Les types de composants sont décrits en utilisant une extension de l'IDL de CORBA qui permet de programmer les composants dans des langages différents et de les exécuter dans des environnements hétérogènes.
Les implantations des composants sont décrites à l'aide du langage de description CIDL (Nano, 2004).

4.4.3 Le modèle Fractal :

Le modèle Fractal (Bruneton et al, 2004) est un modèle à composants académique, il est dédié à la construction, au déploiement et à l'administration (i.e. observation, contrôle, reconfiguration dynamique) de systèmes logiciels complexes, tels les middlewares ou les systèmes d'exploitation. Le modèle a été défini par France Telecom R&D et l'INRIA. Il se présente sous la forme d'une spécification et d'implémentations dans différents langages de programmation. Fractal est organisé comme un projet du

consortium Object Web pour le middleware open source. Les premières discussions autour du modèle Fractal, initiées dès 2000 à France Telecom R&D dans la lignée du projet Jonathan, ont abouti en juin 2002 à la première version officielle de la spécification et la première version de Julia, qui est l'implémentation de référence de cette spécification.

5 Approche orientée service :

L'approche à composant met en évidence certaines limites telles que, la gestion du dynamisme des composants d'une application à l'exécution et le fort couplage entre ses composants.

Pour faire face à ces limites, l'approche à service est apparue. Cette approche utilise les services comme la brique de base pour créer des applications (Huhns et al, 2005), elle propose des moyens pour supporter le développement d'applications flexibles et faciliter l'évolution indépendante des éléments constituant l'application.

5.1 Notion de service :

La notion de service constitue le concept de base de l'approche à service. Dans la littérature, plusieurs définitions sont citées, nous présentons les suivantes :

"Services are self-describing, platform-agnostic computational elements…." (Papazoglou,2003).

"Services are autonomous, platform-independent entities that can be described, published, discovery, and loosely coupled in novel way."(Papazoglou et al, 2007).

Dans ces définitions, *(Papazoglou, 2003) et (Papazoglou et al,2007)* , l'auteur présente un service comme une entité logicielle possédant une description clairement spécifiée et qui est indépendante de sa plateforme d'exécution. De cette manière, un service ne connaît pas son contexte d'utilisation. En plus, il peut être mis à disposition par un fournisseur et un utilisateur peut l'utiliser en ayant connaissance uniquement sa description, sans connaître réellement les détails liés à la technologie utilisée lors de sa réalisation, ce qui offre un couplage faible entre les fournisseurs et utilisateurs de services.

« A service is a mechanism to enable access to one or more capabilities, where the access is provided by using a prescribed interface and is exercised consistent with

constraints and policies as specified by the service composition. A service is accessed by means of a service interface where the interface comprises the specifics of how to access the underlying capabilities. There are no constraints on what constitutes the underlying capability or how access is implemented by the service provider. A service is opaque in that its implementation is typically hidden from the service consumer except for (1) the information and behaviour models exposed through the service interface and (2) the information required by service consumers to determine whether a given service is appropriate for their needs. » (Oasis, 2006).

Pour le consortium OASIS (Oasis), un service fournit un ensemble de fonctionnalités décrites dans une spécification, appelée interface, ainsi qu'un ensemble de contraintes et de politiques d'accès aux fonctionnalités offertes. L'implantation du service n'est pas visible pour l'utilisateur. Seules les informations qui peuvent permettre de savoir si le service correspond aux besoins de l'utilisateur sont disponibles. Nous pouvons noter qu'il est cependant difficile de discerner une information utile d'une information inutile pour le choix d'un service.

(Arsanjani, 2004) apporte une autre définition de service. A la différence des autres, il met en avant les principales interactions qui permettent l'utilisation des fonctionnalités du service :

« A service is a software resource (discoverable) with an externalized service description. This service description is available for searching, binding, and invocation by a service consumer. Services should be ideally be governed by declarative policies and thus support a dynamically reconfigurable architectural style. »(Arsanjani, 2004).

Cette définition s'intéresse à l'utilité de la description du service. Dans un premier temps, le client recherche le service correspondant à ses besoins en fonction de la description de service fournie. Ensuite, il fait la liaison avec lui et il l'invoque. Ces actions peuvent être réalisées avant ou pendant l'exécution de l'application à services.

En résumé, un service possède les caractéristiques suivantes :

- ✓ Un service possède une description rendue accessible par des fournisseurs pour des éventuels utilisateurs.
- ✓ La description d'un service peut contenir l'ensemble des fonctionnalités fournies par ce service, et ainsi que ses propriétés.

- ✓ La description de service ne contient pas les détails liés à la technologie utilisée, tel que le langage de programmation.
- ✓ Un utilisateur de service est capable de rechercher une description correspondant à un service donné pour l'invoquer à tout moment.

5.2 Architecture orientée service :

5.2.1 Définition de SOA (Service Oriented Architecture) :

La notion d'architecture orientée service n'est pas nouvelle, elle est apparue dès le développement des approches client/serveur. Ici encore les définitions sont nombreuses et nous retiendrons les suivantes (Projet RNTL faros, 2006) :

« A service-oriented architecture is a style of multitier computing that helps organizations share logic and data among multiple applications and usage modes » (Shulte et al, 1996).

« Service Oriented Architecture is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations» (Mac Kenzie et al, 2006).

« SOA enables flexible integration of applications and resources by : (1) representing every application or resource as a service with standardized interface, (2) enabling the service to exchange structured information (messages, documents, "business objects"), and (3) coordinating and mediating between the services to ensure they can be invoked, used and changed effectively » (Dodani, 2004).

Malgré le manque de spécification officielle pour définir une architecture orientée services, trois rôles clés sont communément identifiés : producteur de services, répertoire de services et consommateur de services.

5.2.2 Acteurs et interactions dans l'architecture orientée service :

L'objectif de SOA est de fournir des services aux utilisateurs finaux ou aux applications. Cette approche définit une interaction entre les agents logiciels comme un

échange de messages entre clients et fournisseurs. Les agents peuvent être simultanément des clients et des fournisseurs.

La Figure 4 présente les principaux acteurs qui interviennent dans une architecture à services.

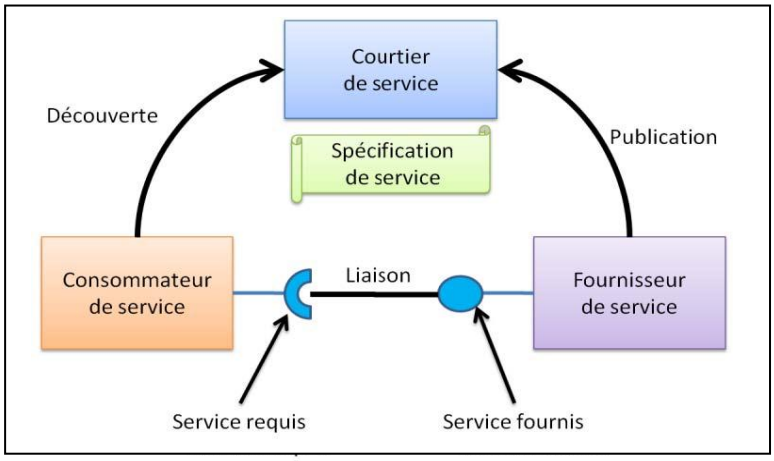

Figure 4 : Acteurs et interactions dans l'architecture à service.

A/ Les acteurs :

L'architecture orientée service est axée autour de trois concepts fondamentaux, à savoir, le fournisseur de services, le consommateur de services, et l'annuaire de publication (O'Sullivan et al, 2002) :

- *Fournisseur de services :* Le fournisseur de services a pour rôle de produire des objets de service qui implémentent une ou plusieurs interfaces de services. Pour supporter la découverte des fournisseurs de service, les descripteurs de services correspondant aux services d'un fournisseur doivent être publiés auprès d'un registre de services (cette publication peut être réalisée soit par le fournisseur, soit par un tiers). Les services d'un fournisseur ont une disponibilité dynamique, ils peuvent à tout moment être ajoutés ou retirés du registre de services.
- *Consommateur de services :* Cet acteur est le client d'un service particulier. Pour pouvoir interagir avec le service, le demandeur de services doit être lié à un fournisseur du service qui doit être découvert auparavant dans le registre de services. La découverte d'un fournisseur dans le registre de services est réalisée à partir de critères relatifs à la description du service qu'il fournit. Lors de la

liaison, le demandeur de services obtient un objet de service de la part du fournisseur.

- *Registre de services (annuaire) :* Le registre de services est l'intermédiaire entre les fournisseurs et les consommateurs de services. Le registre contient un ensemble de descripteurs de services ainsi que des références vers les fournisseurs de ces services. Le registre fournit des mécanismes permettant de l'interroger pour obtenir des références vers les fournisseurs de services.

B/ Les interactions :

Aux trois acteurs de l'architecture orientée service, il a été ajouté trois primitives de communication :

- *Publication de service :* permet aux fournisseurs de service d'enregistrer les services qu'ils fournissent. Cette interaction s'effectue entre les fournisseurs et l'annuaire de service.
- *Découverte d'un service :* permet aux consommateurs de service de rechercher et découvrir les fournisseurs de service qu'ils requièrent. Cette interaction à lieu entre le consommateur et l'annuaire de service.
- *Liaison à un service :* une fois que le service requis est découvert par le consommateur, ce dernier peut se lier à un fournisseur de ce service pour utiliser les fonctionnalités qu'il fournit. Cette interaction s'effectue entre le consommateur et le fournisseur de service.

Une architecture orientée service doit donc mettre à disposition des mécanismes de publication, de recherche (aussi appelée courtage) et d'invocation de service.L'invocation de service doit suivre un contrat défini entre le demandeur et lefournisseur.

(Cervantes, 2004), détaille les différentes interactions d'une architecture orientée services dans la figure 5 :

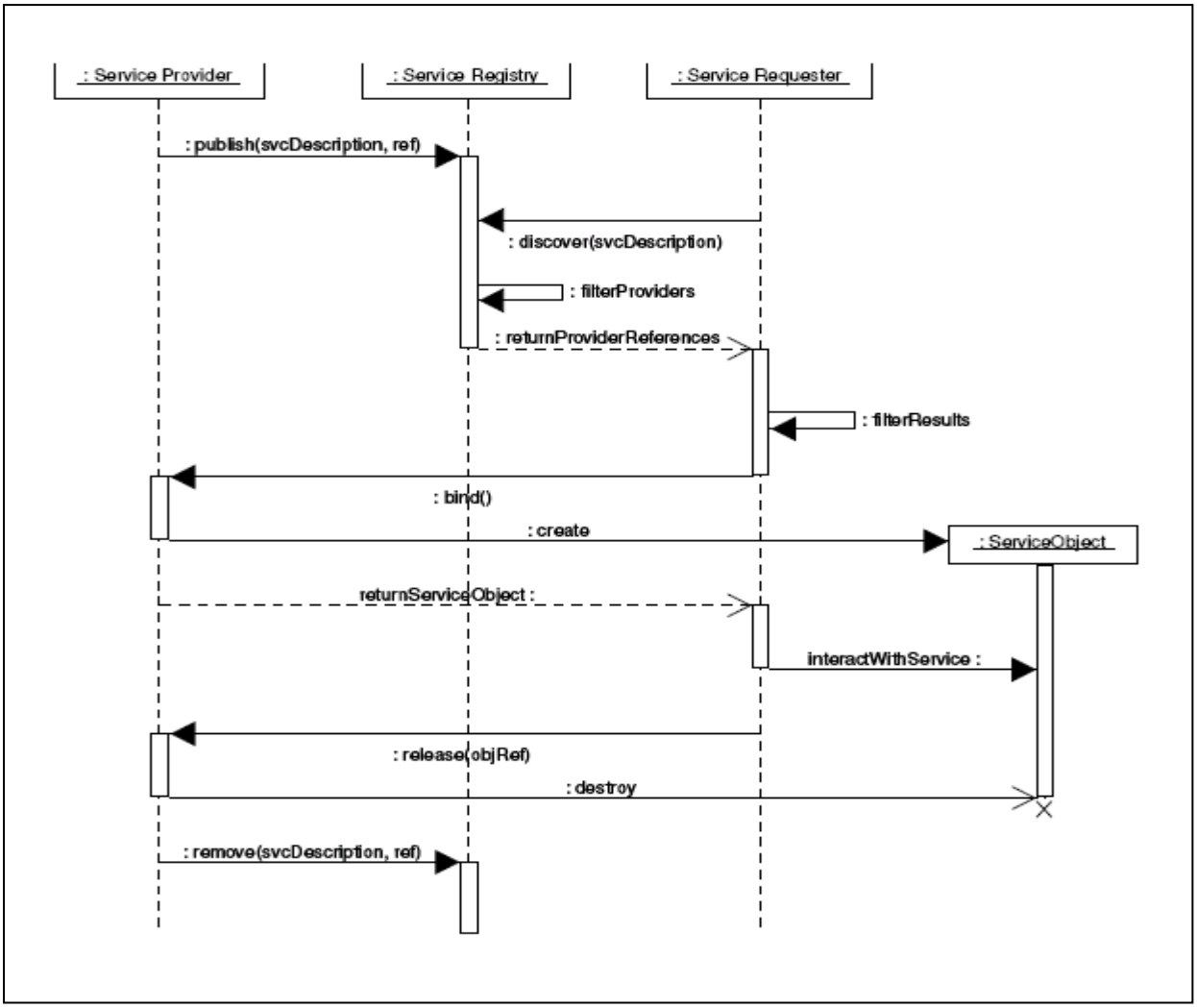

Figure 5: Patron d'interaction d'une architecture orientée services (Cervantes, 2004).

5.2.3 Les caractéristiques de l'architecture orientée service :

- ✓ *Réutilisation :* la SOA offre la possibilité de mettre en œuvre de nouveaux services à partir de ceux existants. Les services sont ainsi réutilisables dans une application.
- ✓ *Le faible couplage :* comme la seule information partagée entre les fournisseurs et les consommateurs est la spécification de service, le couplage entre ces deux entités est faible.

- ✓ *La liaison tardive :* la liaison entre les fournisseurs et les consommateurs a lieu seulement lorsqu'un fournisseur est trouvé et lorsque le consommateur le demande.
- ✓ *Le masquage de l'hétérogénéité :* grâce au faible couplage, un consommateur n'a pas à connaitre les détails d'implémentation du fournisseur de service, ni à sa localisation précise.
- ✓ *Evolutivité et souplesse :* elle offre la possibilité de modifier ou de substituer, à l'exécution, les services utilisés par l'application pour satisfaire par exemple les nouveaux besoins de l'entreprise, supportant ainsi le dynamisme au sein des applications.

5.3 Etude de quelques plateformes à services :

5.3.1 Jini :

Jini (New March, 2001) est la plateforme à services promue par Sun. Basée sur Java, elle permet le téléchargement de code et la sérialisation ce qui en a fait une plateforme distribuée utilisée pour les applications réparties.

Une autre caractéristique de Jini est que consommateurs et fournisseurs de services doivent trouver un registre avant de commencer à interagir. Un registre, qui correspond en pratique à un service de recherche, peut être connu par une adresse fixe ou bien il peut être découvert. Pour cela, les différents acteurs (consommateurs et fournisseurs de service) diffusent une annonce de présence. Cette annonce est écoutée par les services de recherche qui répondent aux annonces et se font connaître. Par la suite, ce sont ces services de recherche qui seront utilisés comme registres.

Les services dans Jini peuvent être organisés par groupes (par exemple groupe services d'impression) et un registre peut héberger un groupe de services particulier. Bien que Jini supporte l'existence de registres multiples, il n'offre pas de mécanismes permettant aux registres de déléguer une demande de service. Au lieu de cela, les fournisseurs de services enregistrent leurs services dans plusieurs registres.

- ➢ *Descripteur de service:* Dans Jini, un service est décrit par une interface Java et un nombre variable d'attributs, qui sont des objets de sous-classes de la classe Entry.

- *Registre:* Il est distribué entre plusieurs services de recherche. Une fois un service de recherche (appelé lookup service) trouvé, le fournisseur de service peut y enregistrer son service. Pour retirer un service, le fournisseur peut attendre la fin du bail ou bien forcer son expiration.
- *Découverte / recherche:* La découverte de services se fait par interrogation du registre qui a répondu à l'annonce. Jini ne propose cependant pas de mécanismes flexibles du côté du registre pour réaliser un filtrage et une sélection plus fine des fournisseurs. Le critère pour déterminer si un service correspond à une demande est que les interfaces du service coïncident avec celles demandées et que les valeurs d'attributs envoyées par le demandeur soient identiques à celles présentes dans la description du service.
- *Liaison:* Elle se fait par le protocole RMI. L'interaction consiste en des appels de méthodes passant par des proxys (appelés « Stubs » et « Skeletons »).
- *Notifications:* Jini fournit des mécanismes de notification asynchrones permettant aux clients d'être informés des changements au niveau de la disponibilité des services.

5.3.2 Services Web :

Le terme service est de plus en plus associé aux services Web (Malloy et al, 2006) qui sont devenus un standard de fait. Ce succès est du au fait que les services Web abordent un domaine où le besoin est très fort, en effet les services Web permettent à des applications hétérogènes s'exécutant au sein de différentes entreprises de pouvoir interopérer à travers des services offerts par les applications. Or l'utilisation de standards tels que XML facilité l'encapsulation et par conséquent l'interopérabilité (Motahari et al, 2006).

Pourtant, bien que la plateforme à services web soit fondée sur les concepts de l'approche à services, l'interaction propre à cette approche n'est pas toujours suivie de façon systématique. Il faut aussi souligner que l'interaction de services dans les services web est réalisée sur une échelle de temps différente des autres plateformes présentées dans cette étude. C'est une des raisons qui fait que certaines organisations n'utilisent pas le registre pour publier leurs services.

La description des services web est réalisée dans un langage appelé WSDL (*Web Services Description Language*). WSDL est basé sur une grammaire XML et permet de décrire l'interface du service, les types de données employés dans les messages, le protocole de transport employé dans la communication et des informations permettant de localiser le service spécifié. Le registre de services, appelé UDDI (*The Universal Description, Discovery and Integration*) supporte l'enregistrement de descriptions de services (appelés types de services) ainsi que de fournisseurs de services (des entreprises).

UDDI fournit des moyens de publier un fournisseur de service (typiquement une entreprise) à travers des pages blanches, jaunes et vertes. Les pages blanches contiennent le nom du fournisseur et sa description, les pages jaunes catégorisent un fournisseur et les pages vertes, plus techniques, décrivent les moyens d'interaction avec un fournisseur: description de services fournis en WSDL ou processus métiers. UDDI est un registre distribué dans lequel l'information est répliquée sur plusieurs sites, en conséquence, un fournisseur de services ne doit publier sa description de service que vers un seul nœud du réseau de registres.

Le protocole de communication par défaut est SOAP (*Simple Object Access Protocol*). SOAP enveloppe le message dans un format XML. Il est la plupart du temps utilisé sur la couche de transport HTTP. L'interaction est schématisée par la Figure 6.

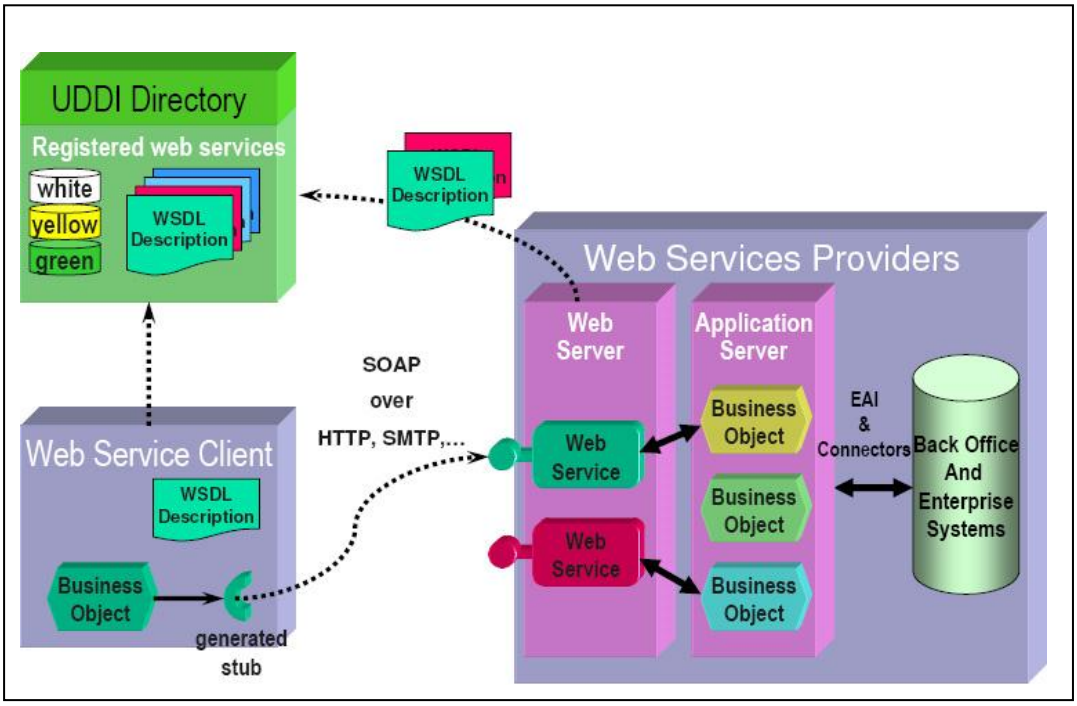

Figure 6: Schéma d'interaction de services Web (Dansez et al, 2001).

La disponibilité dynamique n'est pas intégrée directement à la plateforme des services Web. Comme UDDI n'est pour le moment que très peu utilisé et que sa partie événementielle n'est que peu développée, la découverte de service n'est pas dynamique et le système de notifications n'est que très rarement mis en pratique par les développeurs de services Web. Donc si un service devient indisponible, la réponse à la requête sera une erreur 404 Not Found.

Les services Web sont également utilisés conjointement avec SOAP pour la liaison de composants faiblement couplés dans SCA (Service Component Architecture) (Dansez et al, 2001). SCA est une solution émergeante développée principalement par IBM, BEA et Oracle pour définir un modèle de composant pour les architectures orientées service.

- *Descripteur de service:* Il est écrit dans le langage WSDL basé sur une grammaire XML. Il fournit les informations suivantes:
  - ✓ Définitions : nom du service et espace de nommage
  - ✓ Types : description des types de données complexes
  - ✓ Message : description d'un message. La description contient le nom du message et zéro ou plus parties qui décrivent les paramètres dans le cas d'une requête, ou les valeurs de retour dans le cas d'une réponse.

- ✓ PortType : description d'une méthode formée par un groupe de messages, par exemple une méthode combinant requête et réponse.
- ✓ Binding (rattachement): le protocole de transmission des messages
- ✓ Service : emplacement du service, généralement une URL.

- *Registre:* UDDI est un registre distribué dans lequel l'information est répliquée sur plusieurs sites. Un fournisseur de services n'a par conséquent à publier sa description de service que vers un seul nœud du réseau de registres. L'annuaire UDDI est consultable de différentes manières. Chaque type de pages blanches, jaunes ou vertes donne des informations de nature différentes. Respectivement des informations liées au fournisseur du service (généralement une entreprise), une catégorisation des entreprises et des informations techniques plus précises comme les services fournis et les processus métiers associés.
- *Découverte / recherche:* La recherche se fait donc par interrogation du registre qui permet de trouver le service désiré et d'obtenir son URL.
- *Liaison:* L'usager peut grâce au protocole SOAP interroger le service Web désiré une fois qu'il a récupéré son descripteur.
- *Notification:* UDDI supporte l'enregistrement auprès du registre pour recevoir des notifications concernant des changements dans le registre: ajout, retrait et modification au niveau des services ou des entreprises.

5.3.3 La plateforme OSGI :

OSGi (Osgi, 2010) est un consortium qui a pour but de spécifier, créer et promouvoir une plateforme dynamique à service ouverte / libre. Cette plateforme a pour objectif la livraison et l'administration de services dans des réseaux résidentiels ou autres types d'environnements restreints (automobile, téléphonie mobile, réseaux de capteurs, ...).

La plateforme OSGi offre une surcouche à la machine virtuelle Java. Centralisée / non-distribuée, elle sert d'infrastructure au déploiement de fournisseurs ou demandeurs de service appelés bundles.

Dans la pratique, un bundle est une unité de déploiement qui correspond à un fichier JAR (*Java ARchive*) contenant du code et des ressources. La plateforme OSGi supporte la disponibilité dynamique. En effet elle fournit des mécanismes d'administration permettant de réaliser l'installation, activation, désactivation, mise à jour et

désinstallation des bundles de façon continue. Lorsqu'un bundle est actif, il peut publier ou découvrir des services et se lier avec d'autres bundles à travers un registre de services fourni par la plateforme. Notons qu'un bundle peut à la fois jouer le rôle de fournisseur et d'utilisateur de service, les dépendances de services sont alors décrites dans le descripteur de service.

- *Descripteur de service:* Un bundle contient un fichier *« Manifest »* étendu. Ce dernier fournit des informations sur le fournisseur de service, les services fournis (leur classe d'implémentation), les services requis, sa version, etc.
- *Registre:* Il est centralisé et intégré à la plateforme.
- *Découverte / recherche:* La recherche de services se fait au moyen d'une requête LDAP (*Lightweight Directory Access Protocol*) simple qui peut limiter la recherche à l'ensemble des services qui ont des propriétés spécifiques.
- *Liaison:* Comme la plateforme est centralisée, le consommateur d'un service et le fournisseur sont colocalisés sur la même machine virtuelle Java (JVM). Ils interagissent par le biais d'appels de méthodes Java.
- *Notifications:* La plateforme communique avec ses bundles via un mécanisme événementiel.

5.4 Synthèse :

Pour conclure, l'approche à services est un paradigme qui a pour but de faciliter la construction d'applications à partir de services. Elle favorise ainsi la réutilisation des services existants pour la construction d'applications. L'approche à services aborde spécialement la phase d'exécution de services : la composition d'applications est centrée instance (objet de service).

Une propriété importante de l'approche est le faible couplage entre les consommateurs et les fournisseurs de services. En effet, la seule information partagée entre ces acteurs est la spécification de service, permettant ainsi l'évolution indépendante des services, ainsi que le masquage de l'hétérogénéité et de la distribution des services aux consommateurs.

De plus, l'approche à services permet d'établir les liaisons entre les consommateurs et les fournisseurs de services de façon tardive : la liaison entre les fournisseurs et les consommateurs a lieu seulement lorsqu'un fournisseur est trouvé et lorsque le

consommateur le demande. Grâce aux propriétés de faible couplage et de liaison tardive, l'approche à services permet la construction d'applications flexibles et dynamiques.

Néanmoins, les applications construites par composition de services sont difficiles à administrer. En effet, il existe une dissociation entre le développement et l'exécution d'applications : les informations présentes lors du développement d'une application, notamment les concepts d'implémentation, de dépendance, d'architecture, ne sont pas connus à l'exécution. Du point de vue d'une application, les services sont des entités logicielles « boite noire » et les concepts d'implémentation et de dépendance de services sont inconnus à l'exécution. L'application a donc une vue « utilisateur de services » ignorant leur mise en œuvre, tandis que les composants ont une vue « fournisseur de services » ignorant l'application à laquelle ils participent. Cette séparation de la connaissance entre le développement et l'exécution rend difficile de contrôler et de garantir l'exécution des applications.

L'approche à composants à services vise à résoudre les problèmes liés à la construction d'applications à services en associant les concepts de composant et de service.

6 L'approche à composant à service :

L'approche à services a donné naissance à un nouveau type de modèle à composants, appelé modèle à composants à services (Cervantes, 2004), qui combine les avantages de l'approche à composants : description d'architectures ; et de l'approche à services : faible couplage, dynamisme, afin de faciliter la construction d'applications dynamiques.

L'approche définit un modèle à composants à services qui est un modèle à composants dans lequel les concepts de l'approche à services sont introduits. Cette section décrit les principes de l'approche et présente quelques technologies qui la mettent en œuvre.

6.1 Principes :

L'approche à composants à services propose d'utiliser des composants pour implémenter des services. Cette approche emprunte aux composants l'idée de séparation des aspects fonctionnels et non-fonctionnels, afin de séparer le code de gestion des aspects services (publication, recherche, sélection, liaison, invocation, …) du code implémentant la logique métier des services. Le but est ainsi de déléguer les

mécanismes liés à l'approche à services aux conteneurs des composants, qui interagiront avec l'annuaire de services de la plate-forme sous-jacente.

Les principes de cette approche, décrits dans (Cervantes et al, 2004) sont les suivants :

- Un service est une fonctionnalité fournie. Un service est un ensemble d'opérations réutilisables.
- Un service est caractérisé par une spécification de service qui peut contenir des informations syntaxiques, comportementales et sémantiques, ainsi des dépendances vers d'autres spécifications de services.
- Un composant implémente une ou plusieurs spécifications de service en respectant leurs contraintes. En plus des dépendances spécifiées dans les spécifications implémentées (dépendance de spécification), un composant peut déclarer des dépendances additionnelles vers d'autres services (dépendances d'implémentation). Les services sont ainsi le seul moyen d'interaction entre les instances de composants.
- Le principe d'interaction de l'approche à services est utilisé pour résoudre les dépendances de services. Les services, fournis par des instances de composants, sont enregistrés auprès d'un annuaire de services. L'annuaire de services est utilisé pour découvrir des services de façon dynamique afin de résoudre des dépendances de services.
- Les compositions de services sont décrites en termes de spécifications de services. Une composition (abstraite) est un ensemble de spécifications qui seront utilisées pour sélectionner des composants concrets. Des liaisons explicites ne sont pas nécessaires car elles seront inférées à l'exécution à partir des dépendances déclarées dans les spécifications et les composants participant à la composition.
- Les spécifications de services sont la base de la substitution. Dans une composition, tout composant implémentant une spécification donnée peut être remplacé par un composant alternatif qui implémente la même spécification.

Le modèle à composants à services résultant facilite ainsi la construction d'applications flexibles et dynamiques. En effet, les applications sont définies à un niveau d'abstraction plus élevé, en termes de spécifications de services (en non pas en termes d'implémentations comme dans les modèles à composants traditionnels).

Le principe d'interaction de l'approche à services est utilisé pour résoudre les dépendances de services, permettant de retarder la sélection des implémentations (des fournisseurs de services) jusqu'à l'exécution (contrairement aux modèles à composants traditionnels qui effectuent la sélection avant l'exécution).

Les applications peuvent être définies par une composition comportementale comme l'orchestration, mais aussi par une composition structurelle. De plus, les applications peuvent être composées statiquement avant l'exécution (comme dans les modèles à composants traditionnels), mais aussi dynamiquement, à l'exécution, et par opportunisme. Entre ces deux extrêmes, il est possible de définir des applications dans lesquelles certaines parties de l'architecture sont définies statiquement, et d'autres sont définies dynamiquement et autorisées à varier pendant l'exécution.

6.2 Avantages et limitations de l'approche à composant à service :

L'approche à composants à services combine les principes de l'approche à services et de l'approche à composants dans le but de faciliter le développement d'applications à services dynamiques. Elle propose ainsi d'utiliser des composants pour implémenter des services.

L'approche propose de décrire les compositions en termes de spécifications de services. La composition est centrée spécification, permettant ainsi de retarder la sélection d'implémentations de services jusqu'à la phase d'exécution, et aussi la substitution dynamique d'implémentations de services. De ce fait, l'approche fournit un plus grand degré de flexibilité pour la composition d'applications dynamiques. Cependant, les modèles à composants à services actuels n'implémentent pas tous les principes de l'approche, limitant la construction et/ou l'administration des applications.

7 Avantages et limites des différentes approches étudiées :

Les approches à composants, à services et à composants à services proposent d'assembler des entités logicielles préexistantes dans le but de faciliter et de réduire les

temps et les coûts de développement des applications. Toutefois, ces approches ne permettent pas de créer des applications dynamiques facilement ni de les administrer.
Le tableau suivant détermine les avantages et inconvénients des approches présentées durant ce premier chapitre.

	Avantages	Inconvénients
Approche à composants	Composition structurelle Modèle de développement Reconfiguration	Manque de flexibilité Dynamisme difficile
Approches à services	Faible couplage Liaison tardive dynamisme	Pas de modèle de composition structurelle Administration difficile
Approches à composants à services	Modèle de développement Composition structurelle Reconfiguration Faible couplage Liaison tardive dynamisme Substitution	Modèle de composition et d'administration limités

Figure 7 : Avantages et limites des approches à composants, à services, et à composants à services

8 Conclusion :

Dans ce chapitre nous avons présenté un état de l'art sur les approches à composants et à services, ainsi que l'approche à composant à service.
Ces approches sont actuellement à l'essor pour le développement des systèmes informatiques de grandes tailles et de plus en plus complexes.
La première approche introduit une méthodologie visant à faciliter la construction d'applications à partir de l'assemblage de briques logicielles préfabriquées appelées « composants ».
La seconde approche, celle à service améliore la réutilisation grâce à la réalisation de nouveaux services à partir de ceux qui existent et qui sont disponibles.

La troisième : l'approche à composant à service propose d'utiliser des composants pour implémenter des services.

Et donc, les trois approches ont pour but d'assembler des entités logicielles préexistantes afin de faciliter et de réduire les temps et les coûts de développement des applications.

Chapitre 2 :
L’adaptation dynamique

1 Introduction

L'évolution de l'informatique a ouvert la voie à de nouveaux types d'applications. En effet, l'avènement de l'Internet et l'évolution d'objets informatiques communicants ont permis l'intégration du monde réel avec le monde informatique.

Nous pouvons constater ainsi que les applications modernes sont de plus en plus intégrées dans un monde ouvert en constante évolution : disponibilité dynamique de dispositifs, mobilité et préférences des utilisateurs, changements des besoins et des requis, etc. Cette variabilité des contextes dans lesquels s'exécutent ces applications rend difficile, voire impossible, d'avoir une connaissance complète, à leur conception, des conditions précises dans lesquelles elles seront utilisées et des services qui seraient les mieux adaptés à un instant donné.

Par conséquent, ces applications doivent être définies de manière flexible, permettant leur adaptation aux divers contextes d'exécution auxquels elles seront confrontées et aux évolutions dynamiques de ceux-ci. Ces adaptations doivent permettre aux applications de continuer à fonctionner correctement malgré les variations de leur contexte, et de profiter au mieux des services apparus en cours d'exécution.

Le besoin de construire des applications qui s'adaptent à leur contexte n'est pas nouveau, le défi est ainsi de trouver le dégrée de flexibilité, de dynamisme et d'adaptation requis pour les applications logicielles modernes.

2 Définition de l'adaptation

Plusieurs définitions de l'adaptation ont été citées dans la littérature, nous nous intéressons dans ce qui suit par celles de :

- ✓ Hachette : le dictionnaire universel francophone (Hachette dictionnaire), propose la définition suivante :

« Adapter v. : rendre (un dispositif, des mesures, etc.) apte à assurer ses fonctions dans des conditions particulières ou nouvelles ».

En reprenant la définition sémantique du terme, une adaptation correspond au processus de modification du système, nécessaire pour permettre un fonctionnement adéquat dans un contexte donné. Le terme adéquat signifie que le système correspond parfaitement à ce que l'on attend dans ce contexte précis.

- ✓ (Chaari, 2007) définit l'adaptation comme suit :

« Le terme adaptation est une opération qui consiste à apporter des modifications à une application ou à un système informatique apte à assurer ses performances dans des conditions particulières ou nouvelles pour un contexte précis d'utilisation ».

- ✓ (Senart, 2003) propose la définition ci-dessous :

« Adapter : veut dire modifier tout ou une partie d'un système afin de l'accommoder à son environnement. Il s'agit de prendre en compte statiquement ou dynamiquement le changement de conditions opératoires, les domaines d'application pour lesquels le système s'exécute, et l'évolution du matériel utilisé ».

Et donc, adapter signifie ajuster une chose à une autre. Dans le contexte informatique, on ajuste une application par rapport à des contraintes qui lui sont imposées en fonction de son contexte d'exécution. Dans un système informatique, l'adaptation se fait sur le comportement, la structure ou les paramètres de l'application. Les contraintes quant à elles peuvent être diverses telle que la performance de l'application, l'utilisation des ressources, etc.

Lors de l'adaptation d'une application, il est nécessaire que celle-ci reste dans un état cohérent. C'est pourquoi, il est nécessaire de s'assurer de la validité des stratégies d'adaptation exécutée afin de conserver un système stable.

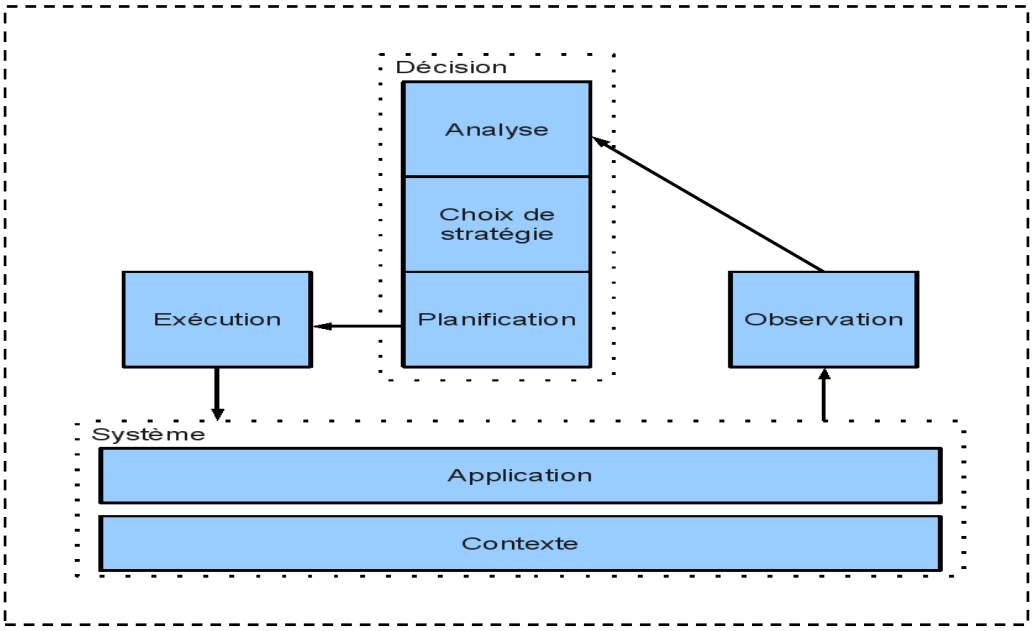

Figure 8 : Représentation de la boucle de contrôle proposée par IBM (Kephart et al, 2003).

Nous distinguons donc, trois phases dans le processus d'adaptation comme il est montré ci-dessus (figure 8) :

La première phase correspond à la phase d'observation. Elle consiste à surveiller le contexte de l'application afin de détecter les changements qui nécessiteraient une adaptation. Ce contexte peut aller des ressources disponibles (bande passante, CPUs, capacité en mémoire) pour le système jusqu'aux préférences des utilisateurs en passant par les propriétés de son environnement (météo, date, . . .).

La seconde phase est la phase de décision. Cette phase est chargée de décider, comme son nom l'indique, de la stratégie à appliquer pour adapter l'application. Cette phase peut elle-même être découpée en deux sous phases qui sont l'analyse et la planification. L'analyse correspond au traitement des informations obtenues et la planification se charge de sélectionner les actions à effectuer pour prendre en compte les changements au niveau du contexte de l'application.

La troisième phase quant à elle, représente l'exécution de la décision c'est-à-dire l'application concrète sur le système des actions qui ont été planifiées.

L'adaptabilité d'un système informatique repose donc sur sa capacité à se conformer à des changements et à des évolutions de son environnement. Les modifications apportées au système peuvent être anticipées avant l'exécution du système dans son environnement (statique), ou peuvent avoir lieu en cours d'exécution (dynamique).

Autrement dit, L'adaptation consiste à rendre un système apte à assurer ses fonctions dans des conditions particulières ou nouvelles. Il existe plusieurs moments pour adapter une application. L'adaptation réalisée plus tardivement (à l'exécution par exemple) est plus puissante (par rapport à une adaptation réalisée au développement par exemple), mais aussi plus complexe à mettre en place et à garantir.

Dans (McKinley et al, 2004), ils proposent une classification d'applications par rapport au moment où les compositions ou les adaptations sont réalisées : développement, compilation, déploiement, exécution (voir la Figure 9).

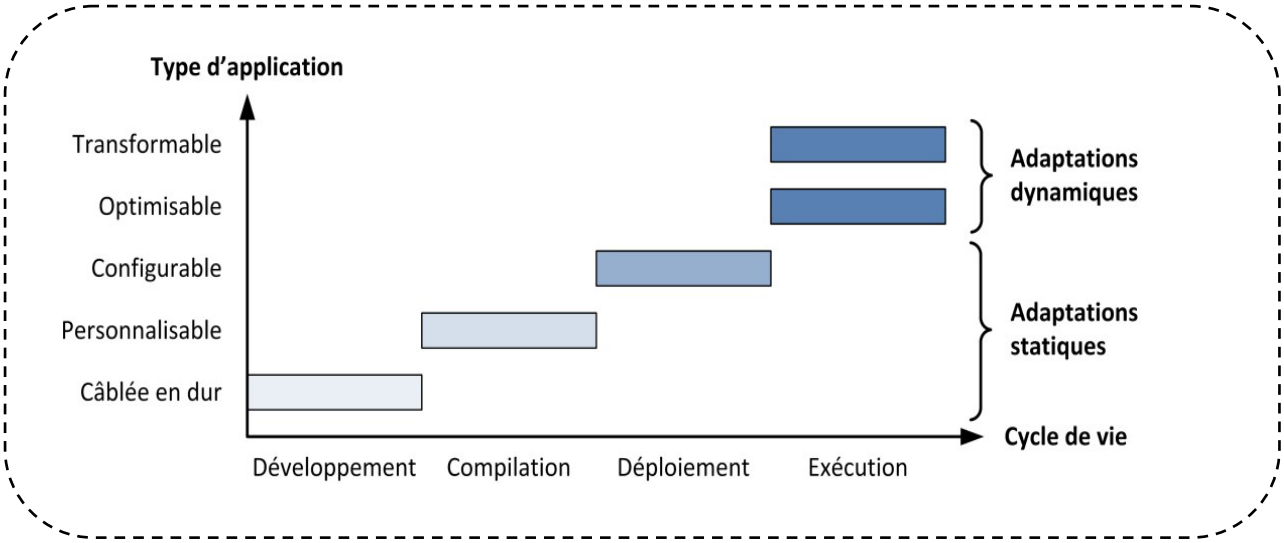

Figure 9: Classification des applications

3 Adaptation statique et adaptation dynamique

L'adaptation peut être réalisée pour corriger le mauvais comportement d'un composant, pour répondre aux changements affectant l'environnement d'exécution, pour étendre l'application avec de nouvelles fonctionnalités ou pour améliorer ses performances. L'adaptation est un événement de plus en plus fréquent dans le cycle de vie d'une application. L'opération d'adaptation doit répondre à plusieurs contraintes comme la sécurité, la complétude et la possibilité de retour en arrière au cas où l'adaptation ne peut se terminer avec succès.

L'adaptation peut être dynamique ou statique, soit respectivement se dérouler à l'exécution ou lors de la phase de chargement ou de compilation d'une application (Chaari et al, 2004).

3.1 Adaptation statique

L'adaptation statique ou manuelle intervient au cours du développement ou avant l'exécution. Elle consiste à adapter manuellement la ressource avant son exploitation dans son nouveau contexte (Kouici, 2002). Cette ressource peut être un fichier (document), une image, un son, une vidéo ou même une application.

Le résultat obtenu par cette technique définit le nouveau contexte de la ressource, ce qui permet de garantir la sûreté de son exploitation.

Cette stratégie présente une solution simple, fiable et efficace, le fait que chaque fournisseur s'occupe de l'adaptation de la ressource pour ses terminaux (Figure 10). Néanmoins, elle présente l'inconvénient de la difficulté d'évolution des applications. Toutes les étapes classiques (la spécification, la conception, l'implémentation et le test) doivent être parcourues au cours de l'adaptation de la ressource, ce qui exige un temps de développement important et par conséquent un facteur critique dans un système d'information.

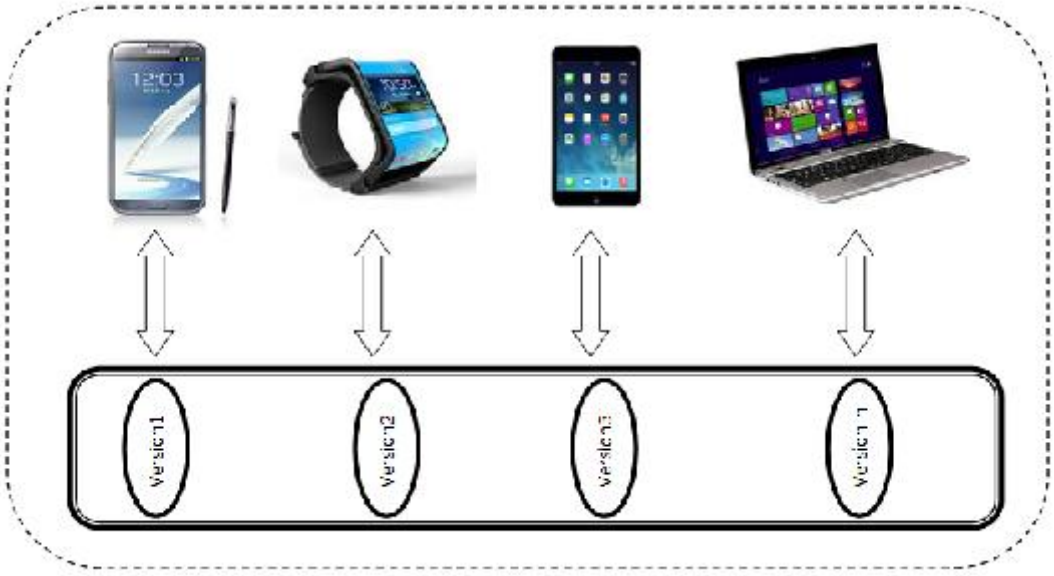

Figure 10: Adaptation statique

3.2 Adaptation dynamique

Contrairement à l'approche précédente, l'adaptation dynamique intervient pendant l'exécution de l'application. Elle consiste à effectuer les transformations sur la ressource au cours de son exploitation (Kouici, 2002). Cette transformation est réalisée suivant les caractéristiques matérielles du terminal demandant la ressource (figure 11).

En effet, cette technique permet de réduire le temps de développement par rapport à l'approche précédente en raison de la génération automatique de la solution suite à la requête de l'utilisateur. Cependant, l'inconvénient réside dans la complexité de passer d'un modèle de description donné à un environnement adéquat pour l'exploitation de la ressource. Elle adresse le problème de diversité des terminaux mobiles.

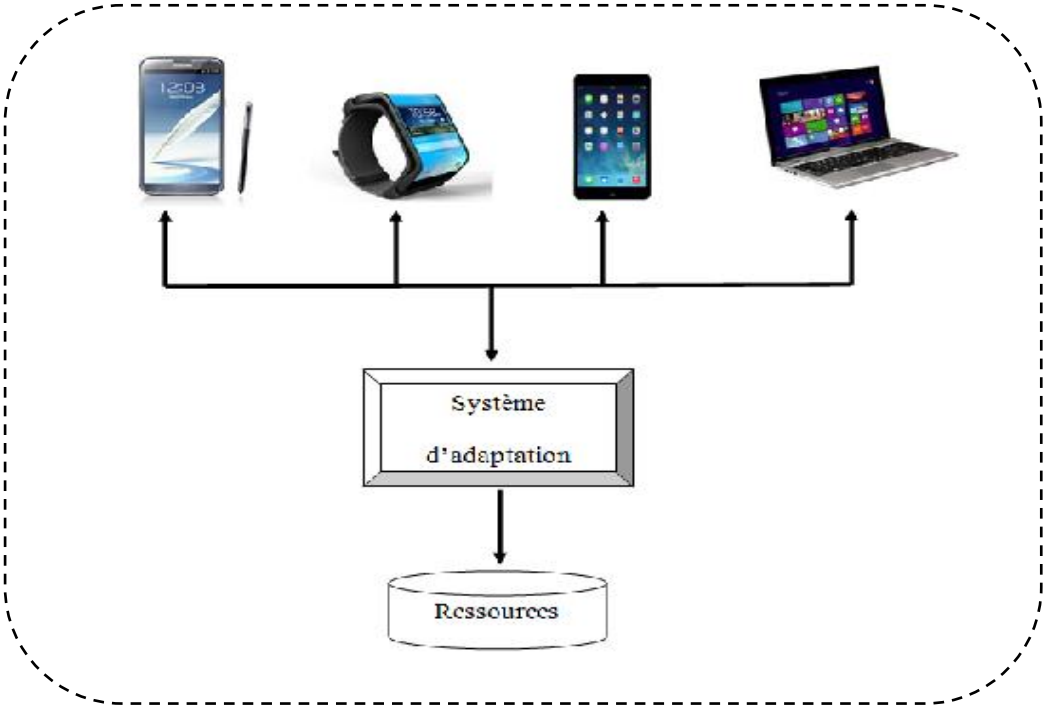

Figure 11: Adaptation dynamique

4 Les raisons de l'adaptation

Dans cette section nous allons parler des raisons pour les quelles une application ou un système doit s'adapter soit vis-à-vis les besoins de le l'utilisateur, ou bien par rapport au contexte dans lequel s'exécute le système.

Après la mise en œuvre d'une application, plusieurs facteurs et raisons interviennent pour l'adapter (statiquement ou dynamiquement). Ces raisons peuvent être divisées en quatre catégories (Ketfi et al, 2002), (Yahiaoui et al, 2004) :

4.1 Adaptation correctionnelle

Si l'application en cours d'exécution ne se comporte pas correctement ou comme prévu. La solution est d'identifier le composant de l'application qui pose le problème et de le remplacer par une nouvelle version supposée correcte. Cette nouvelle version fournit la même fonctionnalité que l'ancienne, elle se contente simplement de corriger ses défauts.

4.2 Adaptation adaptative (évolution technique)

Même si l'application s'exécute correctement, parfois son environnement d'exécution comme le système d'exploitation, les composants matériels ou d'autres applications ou données dont elle dépend changent. Dans ce cas, l'application est adaptée en réponse aux changements affectant son environnement d'exécution.

4.3 Adaptation évolutive ou extensive (évolution fonctionnelle)

Au moment du développement de l'application, certaines fonctionnalités ne sont pas prises en compte. Avec l'évolution des besoins de l'utilisateur, l'application doit être étendue avec de nouvelles fonctionnalités. Cette extension peut être réalisée en ajoutant un ou plusieurs composants pour assurer les nouvelles fonctionnalités ou bien étendre simplement les composants existants.

4.4 Adaptation perfective (optimisation)

L'objectif de ce type d'adaptation est d'améliorer les performances de l'application. A titre d'exemple, on se rend compte que l'implémentation d'un composant n'est pas optimisée. On décide alors de remplacer l'implémentation du composant en question.

5 Les types d'adaptation

Différents types d'adaptation ont été mentionnés dans la littérature. Certaines adaptations concernent les clients de l'application et sont généralement difficiles à réaliser, d'autres affectent l'architecture de conception ou l'architecture de déploiement.
En effet, chaque système logiciel se décompose en objets, composants, services selon son architecture et la technologie utilisée pour sa mise en œuvre. Chacune de ces entités peut être le sujet d'adaptation (attributs, méthodes, composition…).
Dans ce qui suit, le terme «composant » sera utilisé au sens large et ne reflète pas forcement les approches orientées composants.
(Ketfi et al, 2002), (Yahiaoui et al, 2004) présentent cinq types d'adaptation :

5.1 Adaptation des paramètres

Ce type d'adaptation consiste à modifier les valeurs de variables qui influent sur le comportement de l'application sans changer les algorithmes (Zhang et al, 2003), (Haifeng et al, 2002).

5.2 Adaptation de l'implémentation d'un composant

Cette adaptation permet de modifier l'implémentation du composant (le code du composant) sans changer ni les interfaces ni les connexions avec les autres composants de l'application.

D'une autre manière, Ce type d'adaptation touche l'implémentation d'un composant d'une façon individuelle, par exemple, un administrateur d'une application souhaite expérimenter plusieurs algorithmes implémentant la même fonctionnalité, il doit pouvoir passer d'une implémentation à une autre sans arrêter l'application (Capra et al, 2003), (Chen et al, 2001), (Efstratiou et al, 2002).

5.3 Adaptation de l'interface d'un composant

Chaque composant de l'application fournit des services à travers des interfaces bien définies. L'adaptation d'interfaces consiste à modifier la liste des services fournis par un composant. Ceci peut être réalisé en ajoutant/supprimant une interface à/de la liste des interfaces fournies par le composant (Heineman, 2000).

5.4 Adaptation d'architecture de déploiement de l'application (changement géographique)

Une adaptation géographique correspond à la migration d'un composant d'un site à un autre pour la répartition des charges par exemple (Tilevich et al, 2009). Ce type d'adaptation n'affecte pas la structure de l'application, cependant, la communication entre le composant déplacé et les autres composants doit être adaptée selon la nouvelle localisation du composant. L'état interne du composant doit être sauvegardé et réinjecté dans le composant dans sa nouvelle localisation. Les messages reçus et non encore traités doivent être pris en compte.

5.5 Adaptation structurelle (changement de structure de l'application)

Ce type d'adaptation modifie la topologie (architecture globale) de l'application, c'est-à-dire le changement porte sur le changement de la structure en termes de composants et de connexions. Une telle adaptation comporte : l'ajout d'un nouveau composant, la suppression d'un composant existant ou bien la modification des interconnexions entre les composants (Cassagnes et al, 2009), (Dowling et al, 2004), (Garlan et al, 2004), (Layaida et al, 2006).

a/ L'ajout d'un nouveau composant

Ajouter un nouveau composant à une application en cours d'exécution implique l'adaptation des interconnexions entre les composants, en plus le composant ajouté doit prendre en compte l'état de l'application, il doit donc récupérer l'état à partir duquel il doit démarrer et se personnaliser en fonction de cet état.

b/ La suppression d'un composant existant

Supprimer un composant existant ne doit pas affecter l'exécution des autres composants. Il doit être aussi dans un état stable avant qu'il soit supprimé. Il faut aussi veiller pour que les messages et les données échangées avec ce composant ne soient pas perdus.

c/ La modification des interconnexions entre composants

La modification des interconnexions entre composants se fait quand un nouveau composant est ajouté, ou bien l'or d'une modification du comportement de l'application qui implique la redéfinition des interactions inter composants. En général, lorsque deux composants sont connectés, les types de leurs ports connectés doivent être compatibles. Le type de communication (distante ou locale)doit aussi être pris en compte dans une telle modification.

6 Caractérisation de l'adaptation

Après avoir présenté les différentes raisons d'utilisation de l'adaptation dans une application, nous énumérons dans cette section les différentes classifications utilisées dans les travaux existants d'adaptation. Cette classification élaborée par (Chassot et al,

2006) dépend de la manière d'application de l'adaptation : statique, dynamique, centralisée, distribuée, comportementale ou architecturale.

6.1 Statique Versus Dynamique

L'adaptation statique ou manuelle consiste à préparer plusieurs versions de la même ressource avant son exploitation. Cette ressource peut être un document, une image, une vidéo ou même une application. La ressource est adaptée manuellement avant de pouvoir la réutiliser dans son nouveau contexte. Cette technique a été adoptée pour plusieurs développements au début de l'apparition des terminaux mobiles afin de pouvoir exploiter des ressources initialement prévues pour des PC standards. Avec une adaptation statique, le résultat est dédié au nouveau contexte d'utilisation de la ressource ce qui garantit la sûreté de son exploitation. Elle présente aussi une solution simple et efficace car chaque fournisseur s'occupe de l'adaptation de la ressource pour ses contextes. Cependant, elle présente l'inconvénient de la difficulté d'évolution pour la prise en charge de nouveaux contextes d'utilisation.

Contrairement à l'approche précédente, l'adaptation dynamique effectue les transformations sur la ressource au cours de son utilisation. Un système d'adaptation automatique intercepte les requêtes des utilisateurs et effectue à la volée des transformations suivant les caractéristiques du contexte d'utilisation actuel. OPERA (Lemlouma et al, 2002) et MUSA (Menkhaus, 2002) sont des applications où les données sont adaptées dynamiquement aux caractéristiques du terminal utilisé. Elles présentent une adaptation des pages WEB pour différents terminaux en agissant sur leur composition logique, spatiale, temporelle et hypertextuelle. Avec une adaptation dynamique, on gagne un temps de développement important puisque la ressource est transformée automatiquement à la demande de l'utilisateur.

Cependant, des problèmes d'adaptation peuvent surgir dans un nouveau contexte d'utilisation inconnu, en fournissant un résultat d'adaptation qui n'est peut-être pas adéquat à ce contexte.

En outre, le temps d'attente nécessaire à l'adaptation de la ressource peut gêner l'utilisateur.

En effet, l'adaptation dynamique augmente automatiquement la latence de renvoi de l'information demandée par l'utilisateur.

Malgré un grand intérêt pour l'adaptation dynamique, l'adaptation statique reste aussi un moyen sûr et efficace pour assurer l'exploitation des ressources dans différents contextes d'utilisation. Pour assurer une adaptation efficace et ouverte à plusieurs contextes, il est important d'utiliser les deux approches de façon complémentaire.

6.2 Verticale Versus horizontale

L'adaptation peut concerner une ressource locale à une machine ou distribuée sur plusieurs machines (typiquement un protocole). Dans le cas où la ressource est distribuée, le processus d'adaptation peut être centralisé ou réparti sur les différents pairs où les différentes parties de la ressource ont été stockées. Si l'adaptation est centralisée alors elle est qualifiée de verticale, sinon elle est horizontale. La solution centralisée est plus facile à implémenter et donne des performances meilleures au niveau du temps d'accès à la ressource adaptée. Ceci limite l'utilisation de la solution horizontale à des cas particuliers concernant les protocoles de communication. (Bridges et al, 2001) soulève les problèmes de synchronisation de l'adaptation distribuée et propose un protocole dédié pour gérer cette synchronisation.

6.3 Comportementale Versus architecturale

L'adaptation peut être qualifiée de comportementale (ou algorithmique) lorsqu'elle modifie le comportement de la ressource sans affecter sa structure interne. Ceci facilite l'implantation et l'exploitation de l'adaptation. En effet, puisque la structure est la même, la nouvelle ressource reste directement exploitable comme la ressource d'origine. L'implantation est aussi facilitée car cette adaptation n'effectue pas de changements profonds qui demanderaient une réévaluation des modifications effectuées. Cependant, cette approche comportementale limite le degré d'adaptation de la ressource. Elle est très utilisée pour l'adaptation de la couche transport à la qualité de service comme dans (Akan et al, 2004).

L'adaptation est architecturale lorsque la structure interne de la ressource peut être modifiée en fonction des besoins applicatifs de l'utilisateur. Cette solution est intéressante car elle offre un haut degré d'adaptabilité mais elle reste très difficile à implanter. Elle ne reste possible que si on dispose d'une connaissance suffisante sur l'architecture de l'application et ses différents composants. L'adaptation architecturale

reste restreinte aux applications orientées composant. Beaucoup de techniques d'adaptation intéressantes sont proposées pour ce type d'applications.

7 Approche à service dynamique

Dans le chapitre précédent, nous avons présenté l'approche à services et le principe d'interaction entre ses différents acteurs. Comme nous l'avons vu, ce principe d'interaction peut avoir lieu à n'importe quel moment dans le cycle de vie de l'application : au développement, au déploiement ou à l'exécution. De ce fait, les fournisseurs de services peuvent s'enregistrer auprès de l'annuaire et les consommateurs peuvent découvrir les fournisseurs pendant l'exécution des applications.

Le retardement du principe d'interaction à l'exécution permet ainsi la construction d'applications dynamiques. Afin de supporter le dynamisme, deux nouvelles primitives ont été ajoutées au principe d'interaction de base :

- La notification qui informe les consommateurs de l'arrivée ou du départ d'un fournisseur de service.
- Le retrait de services qui signale qu'un fournisseur de service n'est plus en mesure d'offrir son service.

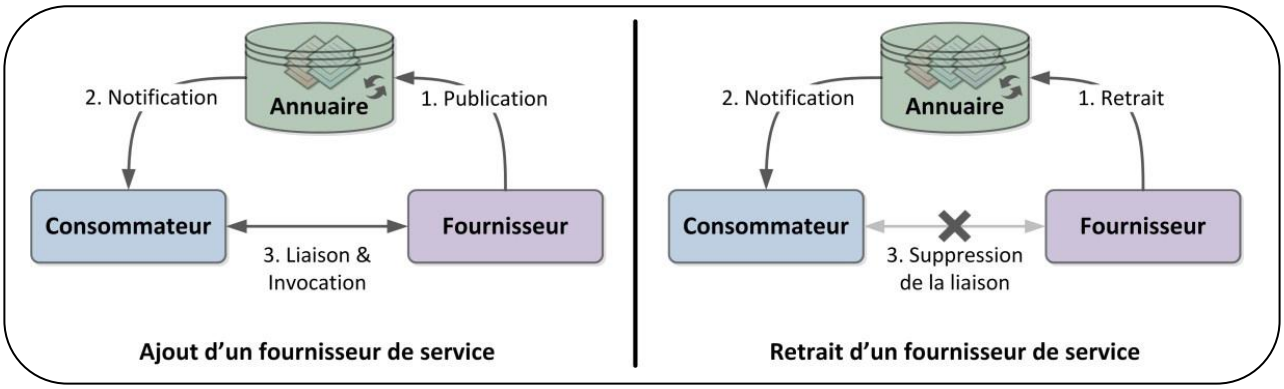

Figure 12: Approche à services dynamique

Grâce à ces deux primitives, les consommateurs de services peuvent gérer la disponibilité dynamique de services et réagir à leur apparition ou leur disparition. Un consommateur d'un service pourrait, par exemple, sélectionner un autre fournisseur d'un service compatible parmi plusieurs disponibles, afin de s'adapter à son contexte d'exécution.

L'architecture à services étendue (SOA étendu) proposée par (Papazoglou, 2003) peut être étendue afin de prendre en compte le dynamisme. Un telle architecture, appelée SOA dynamique étendu, reprend les concepts du SOA étendu, mais se focalise sur la gestion du dynamisme. Ainsi, les primitives de notification et de retrait de services doivent être ajoutées aux mécanismes de base du SOA. Les mécanismes de composition doivent être capables de gérer l'arrivée et le départ des services participant à une composition afin de gérer la structure de la composition et d'assurer son fonctionnement. Les mécanismes de gestion et de supervision doivent être capables de vérifier en permanence la cohérence et la conformité de l'application en fonction des buts. De plus, un SOA étendue dynamique peut fournir des mécanismes additionnels afin de gérer, de manière transversale, des propriétés non-fonctionnelles des applications telles que la sécurité ou la qualité de service. Implémenter tous les mécanismes du SOA dynamique étendu pour la gestion du dynamisme est une tâche extrêmement complexe. De ce fait, la plupart des SOA supportant le dynamisme ne sont pas étendus : ils ne supportent pas la composition ni la gestion d'applications.

Dans le chapitre précédent, nous avons présenté trois architectures orientées service SOA : Jini, OSGi et les services Web. Nous décrivons dans ce qui suit la façon dont OSGI et les services web traitent le dynamisme.

7.1 OSGI

La plate-forme à services OSGi fournit les mécanismes suivants permettant la création d'applications à services dynamiques :

- Des mécanismes de déploiement : les bundles sont déployés et administrés à l'exécution. Il est possible d'installer, de démarrer, d'arrêter, de mettre à jour ou de supprimer des bundles à l'exécution sans arrêter la plate-forme.
- Des mécanismes de notification de services : les primitives de retrait et notification de la disponibilité dynamique de services sont implantées par la plate-forme. Les fournisseurs de services (contenus dans les bundles) peuvent apparaître et disparaître à l'exécution. Un consommateur peut être notifié des arrivées et des départs de services, ainsi que des modifications de leurs propriétés publiées, et agir en conséquence.

Toutefois, comment nous l'avons souligné précédemment, OSGi ne fournit pas de mécanismes de composition d'applications. Toute composition ainsi que le dynamisme qui l'influence doivent être gérés par le développeur. En effet, le dynamisme des services peut valider ou invalider une composition. Le développeur de l'application doit gérer l'apparition et la disparition de services, y compris le relâchement des services disparus.

Ainsi, bien que la plate-forme OSGi fournisse les mécanismes de base pour créer des applications dynamiques, la manque de support pour la définition de compositions rend complexe la gestion des applications au-dessus de cette plate-forme.

7.2 Les services web

Les services Web peuvent être mis à disposition à travers le registre UDDI. Ce registre peut être utilisé pour trouver, au développement ainsi qu'à l'exécution, des fournisseurs et des services. Cependant, le registre UDDI est aujourd'hui peu utilisé : la majorité des organisations qui fournissent des services Web le font sans passer par des registres en indiquant directement leur localisation. Ainsi, la consommation de services entre consommateurs et fournisseurs est directe.

En plus de l'enregistrement et de la recherche de fournisseurs et de services, le registre UDDI permet aux fournisseurs de se retirer de l'annuaire ainsi que de retirer leurs services fournis. Il supporte aussi l'enregistrement de consommateurs de services pour recevoir des notifications concernant des changements dans le registre : ajout, retrait et modification au niveau des services ou des fournisseurs. La gestion du dynamisme (apparition et disparition de services participant à une composition) doit être gérée par les développeurs des applications.

Bien que les mécanismes de sélection et de notification sont spécifiés et implantés, ceux-ci sont rarement utilisés dans le milieu industriel : les entreprises utilisent rarement les annuaires UDDI, préférant assigner aux clients des services Web identifiés et souvent contractualisés.

8 Conclusion

Afin de construire des applications dynamiques et adaptatives, divers aspects doivent être considérés : la spécification du dynamisme autorisé (quand adapter), la

spécification de l'adaptation (comment adapter), l'implémentation de mécanismes supportant la gestion du dynamisme et de l'adaptation (introspection, opérations de reconfiguration, vérification de la cohérence et la conformité).

La mise en place des applications adaptatives reste ainsi très complexe. En effet, la spécification et l'implémentation de la logique d'adaptation de ces applications sont des tâches difficiles qui ont été attaquées par différentes approches. Cependant, nous constatons que les solutions proposées par les travaux existants sont restreintes : elles se focalisent soit sur la spécification du dynamisme et de l'adaptation, soit sur les mécanismes d'adaptation de l'architecture des applications à l'exécution.

Chapitre 3 :

Composition et adaptation de services

1 Introduction

Les approches à composants, à services et à composants à services, que nous avons présenté dans le Chapitre 1, offrent la possibilité de construire des applications en assemblant des entités logicielles préexistantes. Ce processus d'assemblage est appelé une composition de services et le résultat peut être un nouveau service appelé service composite.

La composition de services vise à faire inter-opérer, interagir et coordonner plusieurs services pour la réalisation d'un but (Melliti, 2004).

Deux catégories de services peuvent être distinguées : les services à l'utilisateur et les services logiciels. Les services à l'utilisateur offrent des fonctionnalités qui par leur utilisation facilitent ou aident l'utilisateur dans l'accomplissement de ses tâches. L'adaptation de tels services concerne essentiellement la prise en compte des habitudes utilisateurs, les changements de l'environnement dans lequel évoluent ces utilisateurs. Les services logiciels quant à eux sont des programmes informatiques permettant la communication et l'échange de données entre applications dans des environnements distribués. L'adaptation pour de tels services concerne l'amélioration du processus d'exécution des services (planification, développement et maintenance).

2 La composition de service

2.1 Le processus de composition

L'approche à services spécifie le patron d'interaction entre fournisseurs et consommateurs de services (publication, recherche, liaison, invocation). Cependant, elle n'indique pas comment réaliser une application par composition de services.

La composition de services permet de développer des applications en réutilisant des services existants pour créer d'autres services plus complexes.

En d'autres termes, la composition des services est le processus de construction de nouveaux services à valeur ajoutée, à partir de deux ou plusieurs services déjà présents et existants. Un service est dit composé ou composite lorsque son exécution implique des interactions avec d'autres services, et des changements des messages entre eux afin de faire appel à leurs fonctionnalités. La composition de services spécifie quels services

ont besoin d'être invoqués, dans quel ordre et comment gérer les conditions d'interaction (Arenaza, 2006).

(Bentallah et al, 2002) définit le cycle de vie d'une composition de services à partir de six activités (figure 13) :

- L'encapsulation de services natifs (Wrapping services) : Cette première activité permet de s'assurer que tout service peut être appelé lors d'une composition, indépendamment de son modèle de données, de son format de message, et de son protocole d'interaction.
- L'établissement d'accord d'externalisation (Setting outsourcing agreements) : Cette seconde activité consiste à négocier, établir, et appliquer des obligations contractuelles entre les services.
- L'assemblage de services composants (Assembling composite services) : Cette activité permet de spécifier, à un haut niveau d'abstraction, l'ensemble des services à composer afin d'atteindre l'objectif attendu. Cet assemblage comporte une phase d'identification des services et de spécification de leurs interactions conformément aux descriptions et aux accords entre services.
- L'exécution de services composants (Executing services) : Cette activité consiste en l'exécution des spécifications de la composition précédemment définies.
- Le contrôle de l'exécution de services composites (Monitoring services) : La phase de contrôle permet de superviser l'exécution de la composition en vérifiant, par exemple, l'accès aux services, les changements de statut, les échanges de messages. Ce contrôle permet de détecter des violations de contrats, de mesurer les performances des services appelés et de prédire des exceptions.
- L'évolutivité des services (Evolving services) : Cette dernière phase permet de faire évoluer la composition en modifiant les altérations de l'organisation de services, en utilisant de nouveaux services, ou en prenant en compte les retours de la phase de contrôle.

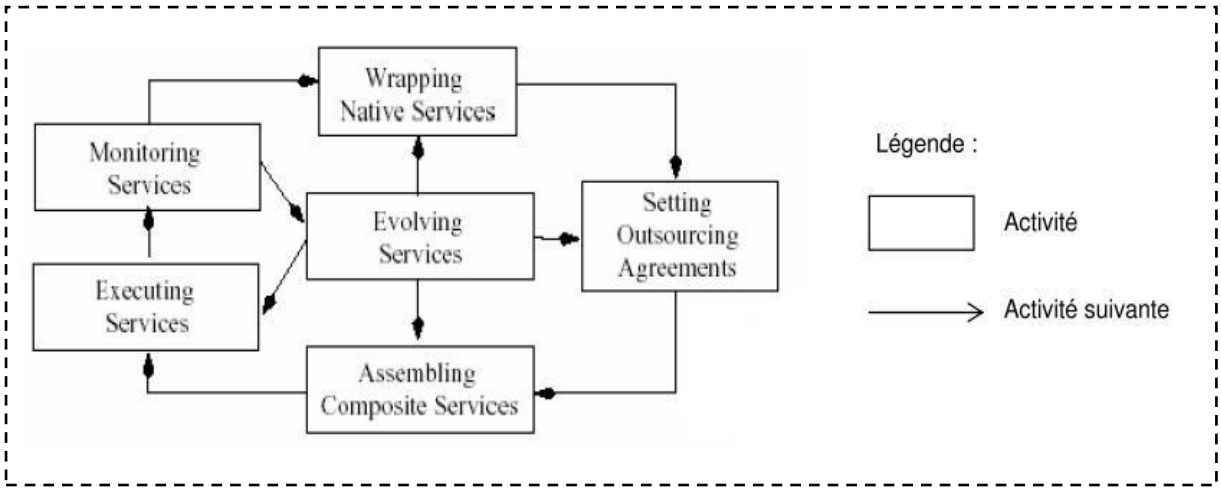

Figure 13: Illustration du cycle de vie de d'une composition de services [(Bentallah et al, 2002).

Le processus de composition de services consiste ainsi tout d'abord à définir un service composite abstrait par les descriptions des services requis, ensuite à découvrir et à sélectionner les services parmi ceux qui sont disponibles, et enfin à réaliser les liaisons entre les services sélectionnés.

Les trois premières phases, c'est-à-dire l'encapsulation de services natifs, l'établissement d'accord d'externalisation, et l'assemblage de services composants (autrement dit découverte, sélection et liaison de services respectivement) peuvent être réalisées avant ou pendant la phase d'exécution caractérisant ainsi la composition. Lorsque tous les services participant à la composition sont sélectionnés et liés avant la phase d'exécution, la composition est appelée *composition statique*. Quand la sélection et la liaison de services peuvent être réalisées à l'exécution, la composition est nommée *composition dynamique*. Enfin, lorsque les sélections et les liaisons déjà réalisées peuvent être modifiés à l'exécution, la composition est appelée *composition adaptative dynamique.*

Malgré sa décomposition en plusieurs phases, le processus de composition de services reste une activité complexe et de longue durée. En effet, chacune des phases est divisée en tâches complexes. Par exemple, les services identifiés pour la composition peuvent présenter des problèmes d'incompatibilité (des types de données d'entrée et de sortie des services par exemple). Des mécanismes de médiation doivent ainsi être crées et utilisés dans la phase de l'assemblage de services (construction) afin d'adapter les services les uns aux autres et de résoudre les problèmes d'incompatibilité. Cela est une

tâche difficile qui, dans la plupart des approches, est effectuée manuellement par les développeurs des applications.

2.2 Les approches de composition de services

Il existe diverses approches pour la réalisation de compositions de services. Elles sont souvent classifiées par rapport à la façon dont le contrôle de la composition est géré : extrinsèque ou intrinsèque aux services. Ces deux façons de gérer le contrôle définissent deux styles de composition : la composition par procédés, aussi appelée composition comportementale, et la composition structurelle.

2.2.1 Approche par procédés (comportementale)

Cette approche, définit la composition de services en spécifiant la logique de coordination de services par un procédé. Un procédé est généralement représenté par un graphe orienté d'activités et le flot de contrôle qui établit l'ordre d'invocation des activités. Chaque activité représente une fonctionnalité réalisée concrètement par un service. La composition est décrite dans un langage spécifique interprété par un moteur d'exécution particulier qui réalise les invocations des services, le routage des donnés d'un service à un autre, et la gestion des erreurs.

Dans une composition par procédé, le flot de contrôle est explicite et le contrôle des invocations de services est externe aux services composés. Nous distinguons deux catégories de composition de services par procédés : l'orchestration et la chorégraphie de services.

L'orchestration et la chorégraphie de services peuvent être combinées pour permettre l'intégration de systèmes d'entreprises différentes. Chaque partenaire modélise en interne, par orchestration, la réalisation des fonctionnalités rendues disponibles. La collaboration entre les différents partenaires est modélisée par chorégraphie.

a/ L'orchestration de services

Plusieurs définitions dans la littérature sont citées, nous présentons les suivantes :

- (Baros et al, 2005) définissent l'orchestration comme un ensemble de processus exécutés dans un ordre prédéfini afin de répondre à un but. Ce type de composition permet de centraliser l'invocation des services Web composants.

Chaque service est décrit en termes d'actions internes. Les contrats entre deux services sont constitués selon le processus à exécuter.

- Tout comme (Baros et al, 2005), (Betallah et al, 2005) définissent l'orchestration comme un processus exécutable, (Betallah et al, 2005) ajoutent que l'orchestration est un ensemble d'actions à réaliser par l'intermédiaire de services. Un moteur d'exécution, un service jouant le rôle de chef d'orchestre, gère l'enchaînement des services par une logique de contrôle. Pour concevoir une orchestration de services, il faut décrire les interactions entre le moteur d'exécution et les services. Ces interactions correspondent aux appels, effectués par le moteur, d'action(s) proposée(s) par les services composants.
- D'après (Peltz, 2003), l'orchestration de services web consiste en la programmation d'un moteur qui appelle un ensemble de services web selon un processus prédéfini. Ce moteur définit le processus dans son ensemble et appelle les services web (tant internes qu'externes à l'organisation) selon l'ordre des tâches d'exécution.

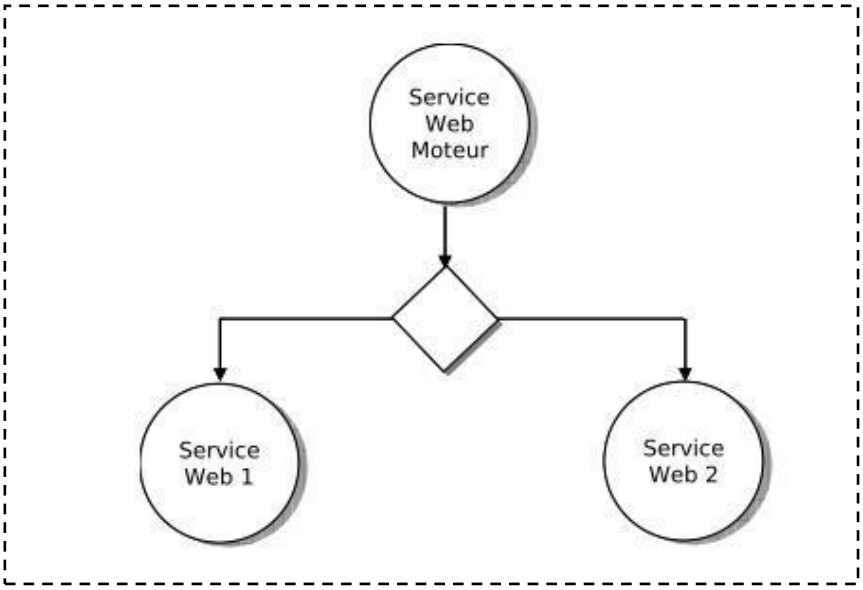

Figure 14: Illustration de l'orchestration selon (Peltz, 2003).

D'une autre manière, l'orchestration de services décrit, du point de vue du service composite, les interactions entre les différentes activités : messages échangés, ordre d'invocation (séquence, parallèle, choix) ; ainsi que des opérations internes réalisées entre ces interactions (transformations de données, invocations de modules internes). La

réalisation de chaque activité correspond à l'invocation d'un service concret. Le contrôle de la composition est centralisé.

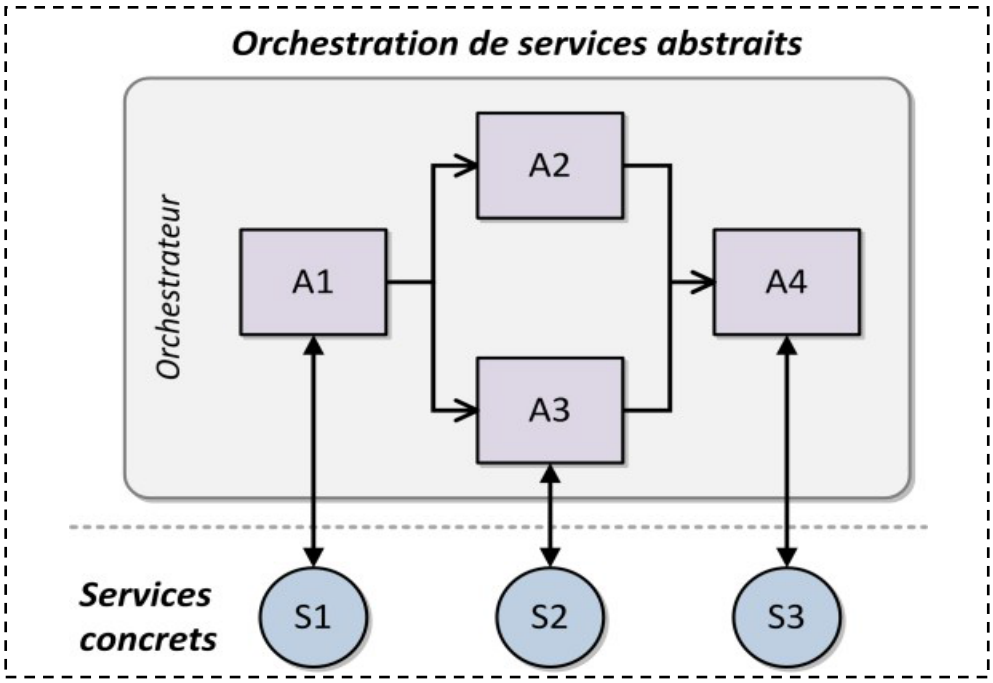

Figure 15 : Orchestration de services

b/ La chorégraphie de services

La chorégraphie de services décrit la collaboration d'un ensemble de partenaires pour atteindre un but commun. La chorégraphie décrit la façon dont les services participants doivent collaborer afin de remplir le but commun : les échanges de messages, les règles auxquelles les interactions entre les différents participants sont soumises, la façon dont les différents participants se coordonnent. Le contrôle de la composition est distribué entre les participants.

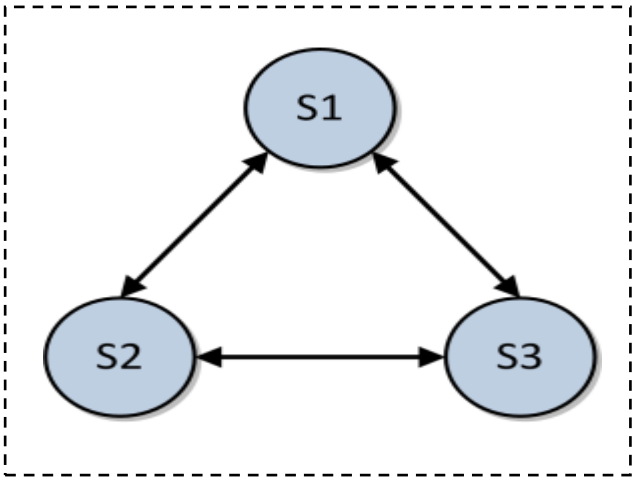

Figure 16: Chorégraphie de services.

Bien d'autres définitions existent, nous citons les suivantes :

- D'après (Baros et al, 2005) la chorégraphie permet de décrire la composition comme un moyen d'atteindre un but commun en utilisant un ensemble de services Web. La collaboration entre chaque service Web de la collection (faisant partie de la composition) est décrite par des flots de contrôle. Pour concevoir une chorégraphie, les interactions entre les différents services doivent être décrites. La logique de contrôle est supervisée par chacun des services intervenant dans la composition. L'exécution du processus est alors distribuée.
- D'après (Peltz, 2003), la description de chaque service Web intervenant dans la chorégraphie inclut la description de sa participation dans le processus. De ce fait, ces services peuvent collaborer à l'aide de messages échangés afin de savoir si tel ou tel service peut aider dans l'exécution de la requête. Chaque service Web peut communiquer avec un autre service Web par l'intermédiaire d'échange de messages. La Figure 20 représente un protocole d'initiation de collaboration entre deux services dans le cadre d'une chorégraphie. Dans cet exemple, le Service Web 1 demande l'exécution d'une méthode du Service Web 2 par l'intermédiaire d'un envoi de message (Requête de service). Cette requête est acceptée par le service Web 2. Ce dernier envoi un message d'acceptation au Service Web 1(Acceptation). Le Service Web 1 accepte le service proposé par le Service Web 2 en lui envoyant un message (Service admis) accordé (Acceptation) par ce second service. Une fois ces messages échangés le Service Web 1 peut invoquer le Service Web 2 dans le cadre de la composition.

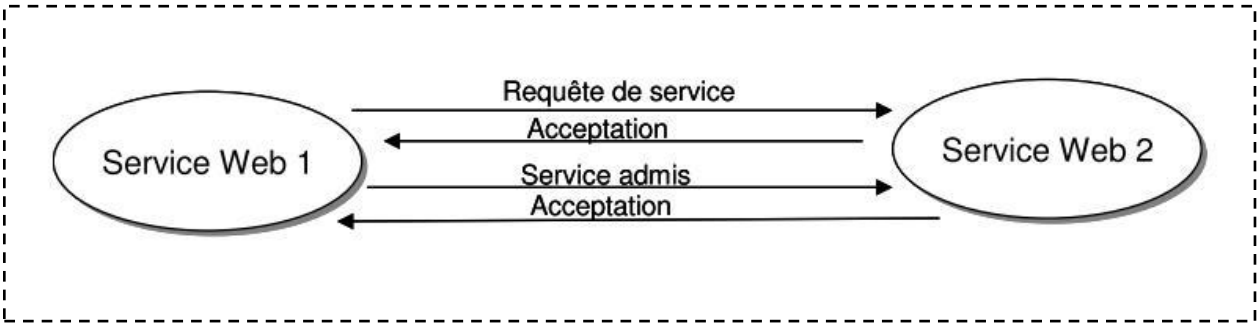

Figure 17: L'illustration de la chorégraphie selon(Peltz, 2003).

La chorégraphie est aussi appelée composition dynamique (Peltz, 2003). En effet, l'exécution n'est pas régie de manière statique comme dans une composition de type

orchestration. Dans une chorégraphie, à chaque pas de l'exécution (à chaque étape de la composition), un service Web choisit le service Web qui lui succède et implémente ainsi une partie de la chorégraphie. La composition de type chorégraphie n'est pas connue, ni décrite à l'avance.

Actuellement, la composition par procédés est utilisée seulement par la technologie des services Web. Divers langages ont été conçus pour l'orchestration et la chorégraphie de services Web comme WS-BPEL (*Web Services Business Process Execution Language*) et WS-CDL (*Web Services Choreography Description Language*) respectivement (voir la section 5).

2.2.2 Approche structurelle

Une composition structurelle consiste à décrire la structure d'une application en indiquant ses composants et les connexions entre eux. Chaque composant déclare explicitement les services qu'il fournit et ceux qu'il requiert. La composition assemble donc des composants dont les services requis par un composant correspondent aux services fournis par un autre composant.

La logique de contrôle, spécifiant comment et à quel moment les opérations des services composés doivent être invoquées, est implicite et repartie entre les différents composants : le contrôle est exprimé à l'intérieur des composants. Par exemple, la Figure 18 montre un assemblage de composants à services dont la logique de contrôle se trouve à l'intérieur du composant A.

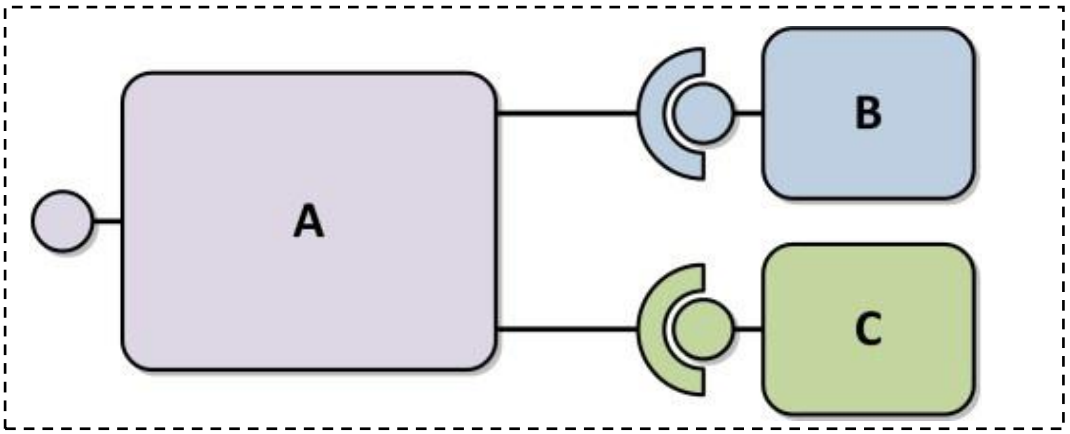

Figure 18: Composition structurelle de services

La définition structurelle d'une composition de services en termes de spécifications de services donne plus de flexibilité à la composition et permet de contrôler l'assemblage des services d'un point de vue global. De plus, d'autres propriétés peuvent être associées à la définition d'un service composite, par exemple pour contraindre la composition (contraintes contextuelles), pour gérer le dynamisme des services, pour adapter la composition, etc.

2.3 Les types de composition des services web

L'idée de la composition de services Web consiste à définit comment les services Web vont être rassemblés selon certains règles, pour atteindre le but demandé par un utilisateur. Une fois que, la description de la composition est réalisée, il est possible de savoir facilement quels services web appartiendront à cette composition. Les solutions proposées peuvent être classifiées selon deux axes :

2.3.1 En fonction du degré de participation de l'utilisateur dans la définition du schéma de composition (Charif et al, 2007)

Les compositions sont soit manuelles, semi-automatiques ou automatiques :

- La composition manuelle : la composition manuelle des services Web suppose que l'utilisateur gère la composition à la main via un éditeur de texte et sans l'aide d'outils dédiés.
- La composition semi-automatique : les techniques de composition semi-automatiques sont un pas en avant en comparaison avec la composition manuelle, dans le sens qu'elles font des suggestions sémantiques pour aider à la sélection des services Web dans le processus de composition.
- La composition automatique : la composition totalement automatisée prend en charge tout le processus de composition et le réalise automatiquement, sans qu'aucune intervention de l'utilisateur ne soit requise.

2.3.2 En fonction de la sélection des services Web

La composition des services web peut être faite au moment la construction du service lui même et est dite *statique*, comme elle peut se faire à la volée en fonction des besoin de l'utilisateur, et dans ce cas elle est qualifiée de *dynamique* :

- La composition statique des services web : dans cette approche, les services web à composer sont choisis à l'heure de faire l'architecture et le design. Les composants sont choisis et reliés ensemble, avant d'être compilés et déployés (Dustdar et al, 2005). Ceci peut marcher en autant que l'environnement des services web, les partenaires d'affaires, ainsi que les dits composants changent peu ou pas du tout. *Microsoft Biztalk* et *Bea Weblogic* sont deux exemples de moteurs de composition statique de services web (Sun et al, 2003). Si les fournisseurs de services proposent d'autres services ou changent les anciens services, des incohérences peuvent être causées, ce qui demanderait un changement de l'architecture du logiciel, voire de la définition du processus et créerait l'obligation de faire une nouvelle conception du système. Dans ce cas, la composition statique des services web est considérée trop restrictive: les composants doivent s'adapter automatiquement aux changements imprévisibles (Sun et al, 2003).
- La composition dynamique des services web : Dans cette approche, les services sont sélectionnés et composés à la volée en fonction des besoins formulés par l'utilisateur (Osman et al, 2005). Cette approche offre le potentiel de réaliser des applications flexibles et adaptables, en sélectionnant et en combinant les services de manière appropriée sur la base de la requête et du contexte de l'utilisateur. Ce type de composition peut engendrer de nombreuses applications utiles, qui n'ont pas été prévues à l'étape de conception. Par conséquent, la composition dynamique de services Web est propice dans un environnement, tel que le Web et l'informatique pervasive où les composants disponibles sont dynamiques et les attentes des utilisateurs variables et personnalisées.

2.4 Les langages de composition des services web

De nombreux industriels (tels qu'IBM ou Microsoft), et consortium (tel que le W3C) travaillent afin de mettre en œuvre un langage de composition de services Web standard tel que WSCI –Web Service Choreography Integration (Arkin et al, 2002). Dans cette section, nous étudions les langages qui sont soit largement utilisés dans l'industrie WS-BPEL (Andrews et al, 2003), soit en cours de standardisation WS-CDL (Kavantzas et al, 2005).

2.4.1 WS-BPEL (Web Service Business Process Execution Language)

BEA (Leymann, 2001), IBM (Thatte, 2001), SAP (Wynen, 2003), Siebel Systems (Wohed et al, 2003) et Microsoft (Kang et al, 2007) ont uni leurs efforts afin de produire un langage de composition de services Web, conçu pour supporter les processus métier à travers les services Web. Ce langage, WS-BPEL (Web Service Business Process Execution Language), est issu de la fusion de deux langages : WSFL – Web Service Flow Language (Kang et al, 2007) d'IBM et XLANG (Reinhard, 1994) de Microsoft. WS-BPEL combine les caractéristiques d'un langage de processus structuré par bloc (XLANG) avec ceux d'un langage de processus basé sur les processus métier (WSFL).

WS-BPEL (Andrews et al, 2003) est basé sur XML et sur les Workflows. Ce langage distingue les processus abstraits des processus exécutables :

- Le processus abstrait : ce processus spécifie les messages échangés entre les différentes parties (services Web composants) sans indiquer le comportement de chacune d'elles. (Saowanee, 2008) parle de Business Protocol, c'est-à-dire la spécification du comportement des partenaires par rapport aux messages échangés, sans rendre public le comportement interne. Ce processus abstrait peut être relié à une composition de type chorégraphie. Les services Web communiquent alors à l'aide d'échanges de messages (partie gauche de la Figure 19).
- Le processus exécutable : ce processus permet de spécifier l'ordre d'exécution des activités, le partenaire concerné, les messages échangés entre ces partenaires, et les mécanismes des erreurs et des exceptions. En d'autres termes, il s'agit du moteur de l'orchestration donnant une représentation indépendante des interactions entre les partenaires.

La Figure 19 illustre la mise en œuvre des deux types de processus (exécutable et abstrait) par WS-BPEL. La définition du processus métier est donnée par le processus exécutable. Les processus abstraits gèrent les invocations entre les différents services Web permettant l'exécution de la composition de services définie dans le processus exécutable.

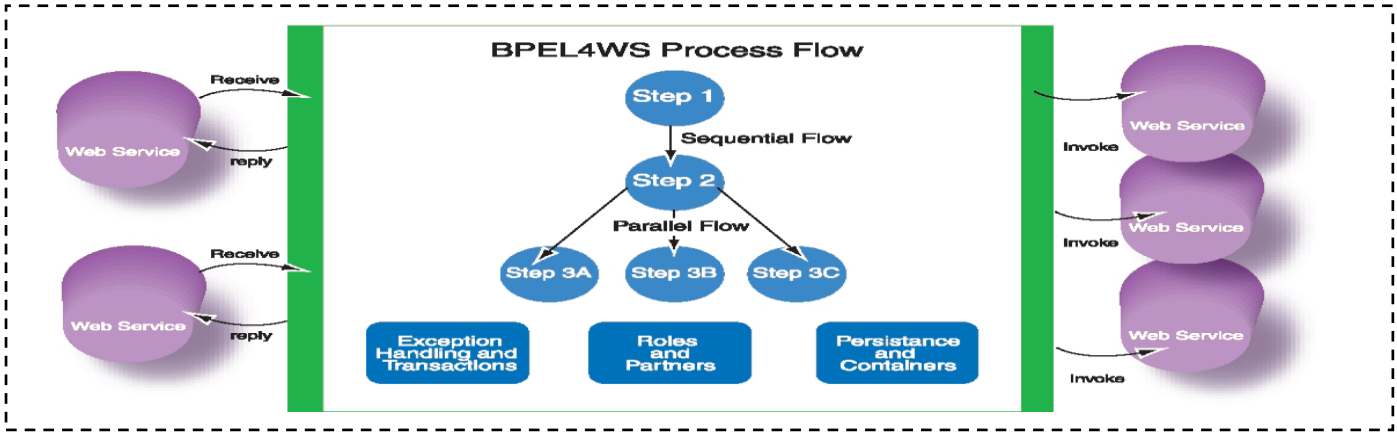

Figure 19: Le flot de processus avec WS-BPEL, d'après (Peltz, 2003).

Trois éléments permettent à WS-BPEL de gérer le flot de processus dans le processus exécutable : les transactions (Exception handling and transactions), les partenaires (Roles and Partners) et les espaces de stockage (Persistance and Containers) (voir Figure 19) :

- Les transactions :Les transactions sont utilisées dans WS-BPEL pour gérer les erreurs et les appels d'autres services si le service appelé est indisponible ou défaillant.
- Les partenaires :Les partenaires sont différents services Web invoqués dans le processus. Ils ont chacun un rôle spécifique dans un processus donné. Chaque partenaire est décrit par son nom, son rôle (en tant que service indépendant), et son rôle dans le processus.
- Les espaces de stockage :Les espaces de stockage permettent la transmission des données. Le flot de processus WS-BPEL permet que ces données soient cohérentes à travers les messages échangés entre les services Web. Un message peut être un message d'appel (invoke), de réponse (reply) ou d'attente (receive).

BPEL contient deux ensembles de constructeurs : constructeurs primitifs et constructeurs structurels. Ces constructeurs permettent la description d'un processus d'affaire exécutable.

Les constructeurs primitifs permettent des fonctionnalités. Parmi ces constructeurs on trouve :

- ✓ <Invoque> : invoque un service Web.
- ✓ <Receive> : attends qu'un client ou un service invoque le business process par envoie de requête.
- ✓ <Reply> : génère une réponse synchrone.
- ✓ <Assign> : permet de manipuler les données des variables.
- ✓ <Throw> : indique les erreurs et les exceptions.
- ✓ <Wait> : bloque l'exécution pour un temps donné.

Les constructeurs primitifs peuvent être combinés pour définir un algorithme complexe spécifiant les étapes par lesquelles passe le business process en utilisant les constructeurs structurels tel que :

- ✓ <Sequence> : définit un ensemble de constructeurs invoqués par ordre d'apparition dans la séquence
- ✓ <Flow> : définit un ensemble de constructeurs invoqués en parallèle
- ✓ <Switch> : pour le choix d'une branche à exécuter
- ✓ <while> : définit une boucle.

WS-BPEL est le premier langage de composition de services Web adopté par la communauté des services Web. Ceci est principalement dû au fait que ce langage possède une grande expressivité dans la définition du processus exécutable (Beth, 1994). L'inconvénient principal de WS-BPEL est que la définition du processus est rigide. Si une activité du processus échoue, le processus dans son intégralité échoue. Aucun retour en arrière et aucune alternative au processus ne sont possibles. De même, lors de la description du processus exécutable, la définition des flots de données ne permet pas de connecter des services dont les entrées/sorties ne correspondent pas exactement. WS-BPEL ne prévoit pas de mécanisme de transformation de données.

2.4.2 WS-CDL (Web Services Choreography Description Language)

Le WS-CDL(Web Services Choreography Description Language) est un langage qui permet de décrire une vision globale des collaborations entre les services, (Kavantzas et al, 2005) insistent sur le fait que WS-CDL n'est pas un langage de description de processus exécutables, ni un langage d'implémentation. Il décrit les séquences ordonnées de messages impliquant plusieurs entités visant à accomplir un objectif

commun.WS-CDL reprend et développe la spécification Web Service Choreography Interface (WSCI).

WS-CDL permet de :

- ✓ Désigner les variables et les types de données échangées.
- ✓ Décrire les activités impliquées.
- ✓ Décrire les structures illustrant les interactions entre les activités.

WS-CDL consiste à définir un fichier XML décrivant une chorégraphie. Les éléments d'un document WS-CDL sont illustrés dans la Figure 20.

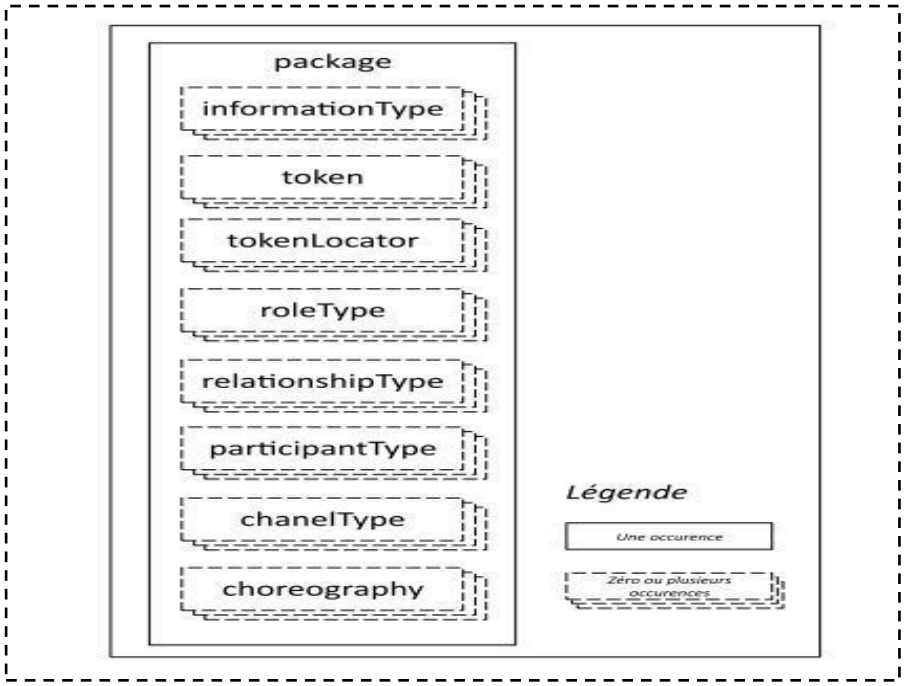

Figure 20: Eléments de description WS-CDL

Nous détaillons l'élément « C*horeography* » décrivant des règles générales d'échange de messages. Il permet de définir trois aspects d'une chorégraphie:

- ✓ Choreography Life-line : décrit une collaboration et indique son état d'avancement,
- ✓ Choreography Exception Blocks : exprime les actions à entreprendre en cas d'erreur,
- ✓ Choreography Finalizer Blocks : indique les méthodes permettant de valider les résultats d'une chorégraphie ou de les annuler.

WS-CDL est un langage de composition de services de type chorégraphie définissant des contrats multi-parties. L'avantage de ce langage est que la description des

interactions (l'élément *Choreography* du Package) est réutilisable. Ceci permet de diminuer la charge de travail des concepteurs. L'inconvénient de WS-CDL est que même si l'élément *Choreography* est réutilisable, sa description reste lourde. Un service doit posséder autant de Package que de participations à des compositions. De plus, WS-CDL n'inclut pas pour le moment de sémantique. Des travaux, tels que (Mc Burney et al, 2008) et (Preuveneers et al, 2009), proposent d'étendre WS-CDL afin d'y intégrer de nouveaux concepts (tels que la gestion des erreurs et des exceptions ou la mise en œuvre d'un minuteur d'exécution de la composition).

2.5 Les défis de composition de Web services

Le problème de la composition de services web est en plein essor. Nous présentons ci-dessous quelques sources de ses complexités :

- la complexité liée à la conception de services web : un Web service n'est pas adaptable. Il est passif jusqu'à ce qu'il soit invoqué. Il a des connaissances de lui-même mais pas de ses applications ou de ses utilisateurs clients.
- le nombre des Web services disponibles est tr`es grand, il est déjà au-delà des capacités humaines pour les analyser manuellement.
- puisque le nombre de Web services augmente jour après jour, il devient plus difficile de trouver artificiellement le Web service qui peut effectuer la tâche à accomplir et encore plus difficile de composer un ensemble de services avec des approches classiques.
- les Web services peuvent être crées et réactualisés à la volée. Par conséquent, le système de composition doit détecter la mise `a jour lors de l'exécution. Ainsi, le schéma de la composition devrait s'adapter en fonction des nouvelles informations.
- les Web services sont généralement établis par des organisations différentes qui utilisent différents modèles conceptuels pour la présentation des caractéristiques des services. Cela exige l'utilisation d'informations pertinentes pour faire correspondre (matching) et composer les services web.

3 L'adaptation de services

Cette section détaille la notion d'adaptation selon différents points de vue de la littérature. Plusieurs concepts s'accrochent à cette notion et il nous semble nécessaire de les présenter pour mieux distinguer et cadrer toute proposition d'une solution d'adaptation automatique de services. Ces concepts sont l'adaptation, la personnalisation, la recommandation et la reconfiguration de services.

Nous présentons à la fin de cette section une synthèse graphique qui illustre les relations entre ces différents concepts.

3.1 Adaptable vs. Adaptatif

Dans (Mc Burney et al, 2008), l'auteur distingue les systèmes adaptatifs des systèmes adaptables. Il appelle système adaptable les systèmes qui permettent à l'utilisateur de modifier certains paramètres de leurs comportements. A l'opposé, il appelle système adaptatif les systèmes qui s'adaptent aux utilisateurs automatiquement en fonction des besoins présumés de ces utilisateurs.

Ceci dit, selon (Papadopoulou et al, 2008), il existe deux procédés d'adaptation de services : l'auto adaptation et l'adaptation contrôlée par l'utilisateur. Dans le processus d'auto adaptation, le système s'adapte sans aucune interaction entre l'utilisateur et le système. Le processus d'adaptation contrôlée quant à lui, laisse à l'utilisateur le soin de prendre la décision, le système automatisant le reste du changement. Selon l'auteur, l'auto adaptation ne fonctionne que dans un nombre de cas limités, qui se caractérisent par une extrême simplicité à cause de la faible expressivité des langages de description des situations nécessaires à l'adaptation.

L'auteur ajoute que même si un service peut s'adapter parfaitement à la hauteur des attentes des utilisateurs, une telle solution est inacceptable pour l'utilisateur, pour des raisons psychologiques. L'utilisateur peut penser que le service n'est pas fiable, à cause de l'absence de pouvoir sur le service et le manque de compréhension des situations où il se sent hors de contrôle, ignorant ce qui s'est passé ou pourquoi cela a conduit à une expérience négative.

(Quan et al, 2006) termine sur une conclusion similaire en disant qu'une auto adaptation peut ne pas être efficace, surtout lorsque l'adaptation ne change pas les actions disponibles à l'utilisateur.

Bien que l'auto adaptation donne plus d'autonomie aux systèmes pour préserver l'intelligence des espaces, l'adaptation contrôlée permet plus de souplesse et de liberté aux interactions utilisateurs dans l'environnement. Toutefois, nous pensons que l'auto adaptation peut couvrir l'adaptation contrôlée si les corrections de l'utilisateur sur les décisions prises par le système sont prises en compte automatiquement dans le processus d'auto adaptation future.

3.2 Personnalisation

La personnalisation selon (Pignotti et al, 2004) est une forme d'adaptation. La personnalisation vise à adapter les fonctionnalités et le comportement du système afin qu'il réagisse différemment en fonction du contexte utilisateur et de ses préférences personnelles. Par exemple, dans (Chaari et al, 2009), les auteurs appliquent un filtrage sur un contenu d'informations pour sélectionner celles qui se réfèrent à un lieu particulier de l'utilisateur.

Selon (Giovani et al, 2005) la personnalisation assure l'adaptabilité du système et facilite la prise en compte des besoins utilisateur. La personnalisation conduit à une adaptation par maintien et construction implicites des préférences utilisateurs en utilisant des mécanismes de surveillance et d'apprentissage appliqués au nom de l'utilisateur. L'auteur (Grodin, 2009) distingue la personnalisation comme étant le processus de création, de maintien et d'application des préférences utilisateurs pour la prise de décision. Par contre, si la personnalisation est combinée avec la prise en compte de l'environnement utilisateur, la décision sera probablement affectée à cause d'un changement de l'environnement qui induira un changement des préférences utilisateur. De ce fait, la personnalisation se fait dynamiquement en accord avec l'évolution et la mise à jour des préférences utilisateur et selon l'état de son environnement.

3.3 Recommandation

Les systèmes de recommandation sont des mécanismes particuliers de personnalisation.

Ils ont pour objectif d'aider l'utilisateur à trouver des informations ou services intéressants. L'idée principale est d'enregistrer les préférences utilisateur afin de proposer des produits ou services supposés intéresser l'utilisateur. L'adaptation dans ce cas de figure est vue comme une anticipation sur les choix de l'utilisateur.

La notion de recommandation est souvent ambigüe dans la littérature. En effet, comme le montrent les travaux décrits ci-dessous, ceci est dû au manque d'une explicitation claire de cette notion. Nous citons ici quelques exemples de ces travaux. Dans (Vallée et al, 2006) les auteurs parlent de recommandation adaptative et mettent l'accent sur la compréhension du contexte comme un acteur majeur dans la conception de ce type d'adaptation. Par contexte, les auteurs veulent dire la date et l'heure, l'emplacement de l'utilisateur, historique des services utilisés par l'utilisateur et les préférences et profil utilisateur. Dans (Cremene et al, 2005) ils introduisent le concept de recommandation personnalisée pour désigner un système de recommandation personnalisée de services. C'est un ensemble d'outils personnalisés qui permettent de trouver des informations ou des services intéressants pour l'utilisateur. Le travail présenté en ((Luis et al, 2004) relie la recommandation à l'adaptation et à la personnalisation. Leur travail consiste à générer automatiquement du code pour l'interface utilisateur afin de fournir des services personnalisés en adaptant les offres, en recommandant certains produits ou en offrant un accès facile à certains services relatifs à la localisation et aux tâches utilisateurs.

3.4 Reconfiguration

La reconfiguration est également une forme d'adaptation. A la différence des adaptations précédentes celle-ci ne prend pas en compte l'utilisateur mais la transformation des paramètres du système selon les conditions internes ou externes au système. C'est par exemple, la réduction du débit de communication en cas de téléchargement intense sur le réseau de la part de l'utilisateur. Dans les travaux de ((Benazzouz et al, 2010) une telle stratégie adaptative est appliquée afin de reconfigurer dynamiquement les services en domotique selon les habitudes d'un nouvel utilisateur et l'état des capteurs. Dans (Ramaparany et al, 2009), les auteurs présentent une autre forme de cette adaptation en prenant un point de vue reconfiguration logicielle, c'est-à-dire qui se conforme à des conditions nouvelles ou différentes. Sa vision est justifiée par le fait que les besoins logiciels sont spécifiques et complexes. A cause de cette

complexité, il peut arriver que les spécifications initiales ne soient plus pertinentes après plusieurs mois d'utilisation de l'application ou du système.

Pour lui, la reconfiguration diffère de l'adaptation si toute situation nouvelle que le système rencontre a été prévue initialement par le concepteur. Toutefois, nous pensons qu'une énumération des différentes situations ou conditions de fonctionnement du système est une tâche difficile voir impossible sauf pour des systèmes à états finis.

Les travaux de (Dockhm, 2007) considèrent une reconfiguration automatique de service en termes d'adaptation au niveau architecture par ajout/retrait/remplacement ou par paramétrage de ses composants. Les auteurs avancent que la plupart des travaux existants qui traitent de l'adaptation dynamique la considèrent dans le cadre des composants logiciels. On cite à titre d'exemple le travail fait dans (Cremene et al, 2006). Cependant, selon ces auteurs les services offrent des possibilités d'adaptation supplémentaires qui ne sont pas traitées dans ces travaux, comme la migration de services.

3.5 Relations entre ces concepts

Cette section discute les relations entre les différents concepts étudiés précédemment afin de préciser ce que nous appelons adaptation automatique dans nos travaux.

Nous avons représenté sous forme schématique (figure 21) le positionnement des différents concepts qui se raccrochent à la notion d'adaptation selon deux axes non orientés. L'axe des abscisses représente les systèmes automatiques selon qu'ils soient adaptables ou adaptatifs, et l'axe des ordonnées représente la source d'adaptation.

Nous retrouvons sur la figure les trois concepts de base discutés dans les précédentes sections: reconfiguration, personnalisation et recommandation. La figure représente l'environnement système comme la principale source d'adaptation pour la reconfiguration au même titre que l'environnement utilisateur pour la personnalisation. Elle distingue par le trait horizontal la part de l'environnement système ou utilisateur prise en compte dans l'adaptation. De plus, le trait vertical sépare les types d'adaptation utilisée quand il s'agit d'automatiser les actions de reconfiguration ou de personnalisation. La recommandation est une forme de personnalisation dont l'objectif est d'anticiper les choix de l'utilisateur. La recommandation s'adresse à un ou plusieurs utilisateurs selon certains paramètres et des préférences qui les concernent. D'où, la

recommandation fera un sous-ensemble de l'ensemble des systèmes de personnalisation. Les services utilisateurs sont embarqués dans l'environnement de l'utilisateur et ils font partie de l'environnement système qui sera pris en compte dans l'adaptation.

Mais l'adaptation automatique de services à l'utilisateur n'est pas une reconfiguration car elle n'a pas vocation à changer le service lui-même : elle implique le dispositif qui donne directement un rendu à l'utilisateur, c'est-à-dire qu'elle a un impact sur le rendu du service pour l'utilisateur. L'adaptation automatique de services à l'utilisateur n'est pas non plus une personnalisation même si elle implique certaines données sur l'utilisateur ou son environnement. De ce qui précède, nous caractérisons l'adaptation automatique de services comme étant une interaction implicite avec le service afin de changer son comportement à l'égard de l'utilisateur.

Les données sur l'utilisateur et son environnement constituent les sources principales de ce changement. Après avoir donné ces particularités, nous considérons à présent l'adaptation automatique dans nos travaux comme un procédé adaptatif assuré de manière automatique et nous pensons qu'elle est du cœur de l'intelligence des systèmes.

Dans ce qui suit, nous nous basons sur cet énoncé pour bâtir les fondements de notre travail.

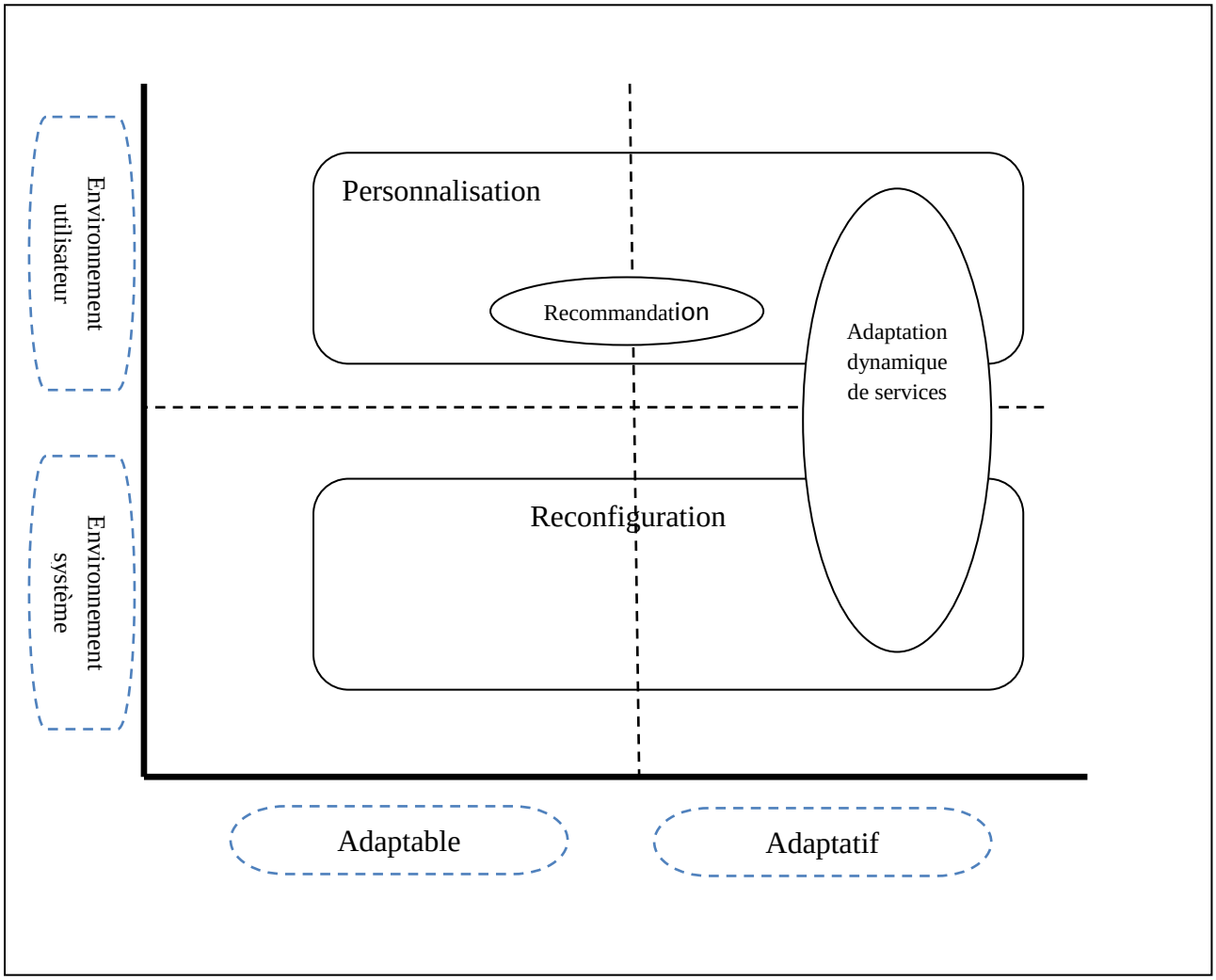

Figure 21: Relation entre les concepts d'adaptation

4 Motivations pour l'adaptation de services

Dans cette section, nous analysons les motivations à l'adaptation de services à l'utilisateur. Ces motivations se résument en deux points : assurer la continuité des services à l'utilisateur et améliorer l'interaction de l'utilisateur avec son environnement.

4.1 Assurer la continuité des services à l'utilisateur

Les technologies actuelles encouragent la prolifération des espaces intelligents pour les personnes allant d'espaces très réduits comme la voiture à des espaces très larges comme une cité ou une ville. Ce développement est soutenu par la miniaturisation continue des composants électroniques et la baisse de leur coût, et est encouragé par

l'émergence de nouveaux besoins utilisateurs comme la réduction de la consommation énergétique, l'aide au domicile des personnes âgées, la surveillance de bâtiments.
L'utilisateur attend de ces espaces un service qui prend en compte ses besoins et assure la tâche demandée. Néanmoins, les services à l'utilisateur sont souvent confrontés à l'absence de mécanismes d'adaptations. Prenons par exemple un service d'appel d'urgence, dans la situation où une personne prépare à manger et utilise une hotte aspirante. A cause du bruit émis, la personne peut manquer un appel urgent. Il est important dans ce cas de trouver une mesure de compensation telle qu'un service d'affichage de message sur une interface déployée dans la cuisine.
Ce genre de solution nécessite des mécanismes d'adaptation qui permettront de basculer de ou vers une solution selon les situations dans l'environnement.
L'adaptation des services à l'utilisateur s'avère un facteur déterminant pour assurer la continuité de ces services et veiller à ce que le service ait bien effectué la tâche demandée.

4.2 Améliorer l'interaction de l'utilisateur avec son environnement

Un des acteurs clés dans la conception des espaces intelligents est d'avoir une interaction plus souple de l'utilisateur avec son environnement. L'adaptation des services à l'utilisateur joue un rôle important dans l'amélioration de cette interaction en facilitant et aidant l'individu dans sa relation avec son environnement physique et social, dans ses dimensions spatiales et temporelles. Par exemple, la diffusion d'un message téléphonique peut être améliorée en tenant compte de l'endroit, la position et l'orientation de l'utilisateur ; l'appel d'aide d'une personne âgée peut se faire de manière automatique selon la disponibilité et le rapprochement des aidants parmi les voisins, la famille et les amis.
Dans (Arentze et al, 2001) ils ont montré qu'en tenant compte des informations de l'environnement, l'adaptation améliore l'interaction utilisateur. Elle permet en effet de répondre à certaines questions telles que "devrais-je éteindre ou diminuer le volume de mon lecteur de musique lorsque le téléphone sonne ?"ou encore "dois-je faire une pause de la chanson en cours si la personne quitte la chambre ?". Cette adaptation a notamment pour effet de minimiser l'attention qu'accorde l'utilisateur aux tâches qu'il

exécute, comme par exemple, l'attention que l'utilisateur doit consacrer à chaque intervention de modification du statut d'un service (Brdiczka et al). L'adaptation peut aider également à anticiper les activités humaines, par exemple, l'allumage du système de chauffage une heure avant que la personne rentre chez elle (Zeinderberg, 2009).

5 Mécanismes d'adaptation de services

La mise en œuvre de l'adaptation de services nécessite des mécanismes afin de préciser quand et comment s'adapter. Ces mécanismes se distinguent en : identification de la situation pertinente pour une adaptation de services, sélection du service à adapter. Le premier vise à déterminer les situations ou les changements dans l'environnement, et le deuxième vise à sélectionner le bon service à activer afin d'agir sur l'environnement ou de changer son propre comportement dans le but de personnaliser, recommander, contrôler des dispositifs, anticiper, simplifier des tâches, etc. Le service peut être associé au préalable à la situation ou au changement, ou sélectionné automatiquement selon un mécanisme bien défini.

Nous analysons dans ce qui suit différents mécanismes. Notre objectif ici, est de voir si ces mécanismes répondent aux motivations présentées auparavant et de déceler les défis résolus ou qui persistent.

5.1 Adaptation à base de règles

L'adaptation à base de règle consiste à écrire les différentes conditions qui régissent la sélection du service sous forme logique en utilisant un langage de description approprié. Il s'agit donc d'un effort important de la part des développeurs. Cependant, il existe des travaux qui tendent à améliorer cette façon de faire. (Smiee, 1992) propose dans sa thèse une solution intéressante pour la définition des règles d'adaptation. Il suggère une plateforme pour les développeurs de services avec un cadre de modélisation, base pouvant être élargie et spécialisée en fonction des applications. L'auteur propose une méthode pour déterminer des situations pertinentes pour l'adaptation de service à partir des spécifications initiales de l'application. A partir des détails sur l'utilisation de l'application, un moteur à base de règles est appliqué pour inférer des situations. Ces situations correspondent à des descriptions de conditions d'utilisations pour une ou plusieurs applications.

Ce travail est intéressant du point de vue conceptuel car il permet de réduire le temps de développement des services et facilite la tâche aux développeurs. Cependant, il est difficile pour ce type de système d'assurer une adaptation de services dans des situations qui n'ont pas été déterminées à partir des spécifications initiales. Ceci impose d'utiliser des descriptions bien expressives pour les spécifications initiales afin de couvrir au maximum les situations pertinentes pour l'adaptation.

De plus, dans une approche d'adaptation à base de règles, les développeurs de services sont obligés de prévoir toutes les situations possibles au moment de la construction des règles qui régissent la sélection des services sachant que le service ne peut pas fonctionner correctement en dehors de ces prévisions (Dietze et al, 2009).

Ceci dit, même si on arrive à prévoir toutes les situations nécessaires à l'adaptation, la construction des règles qui régissent la sélection des services pose d'autres difficultés. D'après (Dockhm, 2007) :

1. Pour chaque service, il est nécessaire d'écrire des règles spécifiques.
2. Si un service doit s'adapter à de nouvelles situations qui n'ont pas été identifiées lors de sa construction, l'intervention du concepteur est nécessaire afin de définir de nouvelles règles où de réécrire le service.
3. Le lien entre une situation et le service qui doit être activé ou déclenché durant cette situation est connue uniquement du développeur du service et non déterminé automatiquement.

Selon (Nayak, 2004), les systèmes à base de règles ont attiré beaucoup d'attention mais leur utilisation comme modèles de prévision dans la planification des activités a été limitée en raison du manque de techniques statistiques permettant de dériver ces règles à partir des données des activités. L'enjeu pour les systèmes à base de ces mécanismes est donc à la fois dans la description des situations de l'utilisateur et de son environnement, et dans leur capacité à maintenir plus de flexibilité et d'ouverture.

5.2 Adaptation à base d'apprentissage automatique

L'apprentissage automatique est l'un des piliers de l'intelligence artificielle par sa capacité d'apprendre à partir de l'expérience. Particulièrement, l'apprentissage par renforcement a pour but de concevoir des systèmes autonomes et adaptatifs.

Pour cela, nous allons parler plus en détails de quelques travaux utilisant ce mécanisme puis nous montrons ses avantages et ses inconvénients.

(Qianhui et al, 2007), emploient un apprentissage supervisé. Le modèle de situation est appris et étiqueté par l'expert, puis l'utilisateur donne un feedback en indiquant le service correspondant durant chaque situation. De manière similaire, (Hheierman et al, 2003) propose que le système détermine le modèle d'adaptation à partir des interactions du système avec l'utilisateur et de l'observation des réactions de l'utilisateur aux réponses du système. Pour cela, elle utilise un algorithme d'apprentissage par renforcement. Elle définit un environnement équipé d'un système ambiant capable de détecter le contexte de l'utilisateur

et d'agir en fonction de cette observation. Un ensemble de prédicats représente un état de l'environnement. La sélection de l'action à entreprendre est basée sur un entraînement de la part de l'utilisateur en utilisant un apprentissage par renforcement avec un entraînement simple de type récompense positive ou négative.

Auparavant, (Raibulet, 2009) a proposé un système similaire pour déterminer le modèle utilisateur de manière implicite, en s'appuyant sur l'historique de l'interaction de l'utilisateur avec le service. Cette nouvelle approche a été validée en utilisant un modèle construit par apprentissage durant l'utilisation d'un service d'information (sport, politiques, finances, etc.). L'auteur utilise les mots contenus dans les articles comme des indicateurs de l'information et construit un réseau de neurones à partir des articles lus ou rejetés durant la session de lecture. Pour les auteurs un mot d'un document peut avoir plusieurs significations selon le sujet abordé ; de ce fait, ils précisent que la sémantique a été le grand handicap de cet ravail car le système proposé ne considère pas l'interprétation exacte du mot dans chaque document.

Du côté des applications commerciales, la solution d'Adecco (société suisse) (Chaari et al, 2007) propose un apprentissage des habitudes du foyer en termes d'utilisation des appareils électriques pour automatiser leur allumage ou leur extinction. L'utilisateur de la solution Adecco a la possibilité d'agir sur les paramètres de configuration et de contrôle localement ou à travers Internet en utilisant une interface logicielle.

En prenant ces données, le système peut apprendre à partir des entrées utilisateur pour automatiser des actions. Par exemple, le système peut apprendre que dans certains cas de figure (caractérisés par des données de capteurs) il faut envoyer une alerte par mail et

non par SMS. Adecco propose un système capable d'être configuré par l'utilisateur pour réagir à des situations déterminées. Les corrections de l'utilisateur dans le temps sont prises en compte par le système pour les futures réactions. Par contre, les données d'entrée pour le système proposé proviennent de l'interface de configuration ou de paramétrage des dispositifs et aussi de capteurs: l'utilisateur intervient pour définir les règles ou les ajuster à ses besoins. Ces règles s'adaptent alors selon les corrections de l'utilisateur dans le temps. Nous mentionnons en dernier que dans ce système, le contrôle des dispositifs n'est pas fait à travers des services au sens informatique.

Ceci dit, un des intérêts pour l'adaptation à base d'apprentissage est la capacité de cette technique à offrir des solutions commerciales. L'avantage le plus clair est que les mécanismes présentés ci-dessus sont moins rigides que ceux présentés dans les sections précédentes, et ce, bien qu'ils nécessitent un entraînement soit pour définir la situation pertinente pour une adaptation de services, soit pour associer la situation pertinente au service à adapter.

(Eheierman et al, 2003) avance un autre avantage en précisant que l'utilisation de ce mécanisme pour l'adaptation dynamique aux situations est possible si on trouve les actions adéquates à chaque situation. Par contre, la technique choisie devra respecter plusieurs contraintes notamment : facilité de l'entraînement car il sera à la charge de l'utilisateur, possibilité d'injecter des connaissances initiales pour que le système ait un minimum de cohérence avec le comportement de l'utilisateur, dès l'initialisation.

Les solutions présentées ci-dessus apportent une amélioration dans l'interaction de l'utilisateur avec son environnement mais elles échouent à assurer la continuité des services si le modèle d'apprentissage rencontre une situation pour laquelle il n'a pas été entraîné. Le temps réel et l'entraînement du modèle restent les difficultés majeures pour ces systèmes. D'un côté, il faut assurer un maximum de fiabilité et de robustesse du système pendant la mise en marche et de l'autre côté si le système est mis à disposition de l'utilisateur novice, cette technologie peut ne pas être maîtrisée.

5.3 Adaptation à base de comparaison

Cette approche est moins utilisée par rapport à l'adaptation à base de règles ou à base d'apprentissage. Elle consiste à comparer les conditions liées à l'exécution d'un service avec les données de la situation en cours. Les conditions sont définies au préalable, par

le concepteur, dans la description des services. Le service qui correspond le mieux à la situation courante est sélectionné même si les données sur cette situation sont incomplètes. (Bhusate et al, 2009) proposent un algorithme basé sur une mise en correspondance des descriptions des services avec les besoins utilisateurs en utilisant des mesures de similarité. Une approche similaire, utilisée en (Chaari et al, 2009), propose une adaptation par comparaison des profils définis dans l'application avec le profil de l'utilisateur. Une fois le profil sélectionné, les règles d'adaptation déjà définies dans le profil sont appliquées.

Dans cette approche, la description sémantique des situations et des services est le paramètre essentiel pour le bon fonctionnement du système. Vu que ces descriptions sont définies par le concepteur, la continuité des services n'est alors assurée que pour les cas prévus.

5.4 Adaptation à base de fouille de données

Actuellement, nous assistons à un intérêt croissant pour les techniques de fouille de données. Celles-ci ont en effet montré leur capacité à fournir de bons résultats dans plusieurs domaines tels que le WEB et l'analyse des transactions d'achats. Pour le cas de l'adaptation des services, l'approche consiste à explorer les données d'utilisation des services en utilisant ces techniques afin d'identifier des situations potentiellement utiles ou des tendances dans les données profitables pour l'invocation future des services. Les solutions discutées en (Jaga et al, 2001), (Serena et al, 2007) concernent la découverte des motifs d'usage du service pour les exploiter dans la détection des services similaires ou la recommandation de certains services parce que les clients ont l'habitude d'utiliser des services du même genre. Les techniques employées sont celles provenant du domaine de la fouille de données ou Data Mining.

6 Défis soulevés par l'adaptation de services

La conception de services adaptatifs supportant des environnements et des besoins utilisateurs dynamiques nécessite la résolution de différents défis. A partir des deux sections précédentes, nous allons les résumer ici.

6.1 Prise en compte du contexte

L'ensemble des situations de l'utilisateur et de son environnement est décrit par le terme général "contexte". Sa prise en compte est essentielle dans l'interaction de l'utilisateur avec les systèmes. Une des questions de recherche qui reste ouverte est de trouver le moyen de percevoir ce contexte. Les solutions actuelles proposent des capteurs divers pour acquérir les données sur l'utilisateur et son environnement mais elles sont hétérogènes et proviennent de sources diverses et variées. Il en découle que les situations nécessaires à l'adaptation des services deviennent très complexes à cerner et conceptualiser.

6.2 Gestion de volumes de données

Les espaces intelligents sont caractérisés par un volume important de données provenant de capteurs et dispositifs. En plus du nombre de capteurs et dispositifs déployés dans l'environnement, le volume de ces données est augmenté parla dimension temporelle de ces applications. En conséquence, des solutions de stockage et d'acquisition sont nécessaires afin de permettre l'exploitation de ces données par des mécanismes d'adaptation.

6.3 Représentation de la sémantique

Les services représentent une grande partie des fonctionnalités offertes par les espaces intelligents. Ils vont devoir exploiter les données sur l'utilisateur et son environnement pour s'adapter. Les appareils, les dispositifs dans l'environnement et les Services Web locaux ou distants constituent les sources potentielles de ces données. Il est donc important pour un système d'adaptation automatique de services d'avoir des descriptions expressives sur les sources et les services car la découverte automatique de nouvelles sources et de nouveaux services exige d'un côté d'avoir l'identité de chacune des sources et ses types de données et de l'autre de connaître les propriétés des services et leurs capacités.

La technologie du Web Sémantique fournit des descriptions souvent complexes et de donner du sens aux termes échangés. Cependant, le défi réside dans la définition de ces

descriptions et la conception de l'ontologie qui prend en compte l'hétérogénéité sémantique des données échangées.

6.4 Extraction de connaissances

Ce défi est reconnu comme étant la capacité d'identifier, de concevoir les connaissances utiles pour le processus d'adaptation et de montrer comment ces connaissances sont exploitées en temps réel, en particulier dans un environnement très dynamique (Schilit et al, 1994). (Pascoe et al, 2000) clarifient ce défi comme étant une création d'intelligence : analyse des informations du contexte, déduction de la signification ou de la compréhension, et intégration de cette connaissance pour l'adaptation des applications. Dans d'autres travaux ce besoin a été exprimé comme de la fusion d'informations ou de la déduction. Par exemple, (David et al, 2005) mentionnent que la fusion des données de capteurs et des actions utilisateur pour l'adaptation est un défi spécifique pour coordonner l'interaction complexe des utilisateurs, des dispositifs, des capteurs et des systèmes intelligents et offrir des meilleurs services.

6.5 Traitement en temps réel

Adapter les applications et les services à l'exécution est très difficile puis que le temps nécessaire entre la détection, la reconnaissance du modèle d'adaptation et le choix ou la sélection des actions à entreprendre est très court (Chaari et al, 2009).

6.6 Ouverture et flexibilité

Avoir des architectures d'adaptation ouvertes et flexibles est un grand enjeu de recherche. (Tigli et al, 2006) argumentent qu'il n'existe pas d'architecture ouverte, flexible et évolutive d'adaptation. Cela pose aussi la question d'étendre les applications et services existants pour supporter l'adaptation. (Chaari et al, 2006) mentionnent que l'écriture des applications d'adaptation demeure une tâche ardue pour les développeurs d'applications. Les approches existantes demandent la conception de toutes les pièces d'adaptation au moment du design.

7 Conclusion

Ce chapitre a permis de préciser les notions de composition et d'adaptation de services. La composition de services désigne une interaction entre deux ou plusieurs services en vue d'accomplir des objectifs déterminés.

Autrement dit, Si l'objectif du concepteur d'une application n'est pas atteint par l'invocation d'un simple service Web élémentaire, alors le concepteur doit combiner les fonctionnalités d'un ensemble de services.

L'adaptation quant à elle, a un intérêt particulier quand il s'agit de l'utilisateur par sa capacité à lui offrir plus de liberté et de souplesse dans ses interactions et son apport dans la robustesse aux changements de l'environnement. Nous définissons alors l'adaptation de service à l'utilisateur comme étant la capacité à faire bénéficier le service de l'expérience de son utilisation en tenant compte des situations de l'utilisateur etde son environnement.

La notion de contexte qui se réfère aux situations de l'utilisateur et de son environnement apparaît comme l'enjeu principal d'adaptation des services à l'utilisateur par rapport aux défis présentés ci-dessus. En effet, la difficulté réside dans la détermination des situations utiles ou pertinentes pour l'adaptation dans un environnement dynamique difficile à caractériser. En faisant le lien entre le contexte et l'adaptation automatique, il nous semble clair que l'interaction de l'utilisateur avec son environnement à travers des services sensibles au contexte et adaptatifs est une approche prometteuse qui mérite d'être creusée.

Chapitre 4 :
La notion de contexte et contribution

1 Introduction

Un logiciel est exécuté dans les infrastructures informatiques complexes, hétérogènes, et très entrelacées dans lequel une diversité d'événements peut se produire. Par exemple, les menaces de sécurité, problèmes de réseau, la réduction de la performance dans l'un des serveurs, etc. Dans ces situations, il est souhaitable d'adapter le logiciel pour continuer à offrir la fonctionnalité requise. L'adaptation du logiciel peut être considérée comme la possibilité pour l'homme de reconfigurer le logiciel puis le redémarrer, ou la capacité du logiciel à se reconfigurer en cours d'exécution (Jaeger, 2007). Le premier cas d'adaptation est considéré comme statique, quand au second il représente l'adaptation dynamique. Il est possible d'effectuer des adaptations statiques dans les cas où le système peut être arrêté afin de faire les adaptations manuelles requises. Cependant, il existe des systèmes critiques qui ne peuvent pas être arrêtés pour mettre en œuvre les adaptations, par exemple logiciels qui exécutent des réseaux électriques et des logiciels pour la banque mondiale. Dans de tels cas, le logiciel a besoin d'adapter dynamiquement son comportement lors de l'exécution en réponse à l'évolution des conditions dans son infrastructure informatique de support (Rosenberg, 2009), (Coppolini et al, 2007), (Colombo et al, 2006).

Ces logiciels ou bien applications sont de plus en plus construits à base de services et exécutés dans des environnements large échelle, dynamiques et distribués.

Le fonctionnement d'une application à base de services doit tenir compte de différents éléments qui interagissent avec son fonctionnement, à savoir :

- Une application de service est généralement construite par assemblage d'une grande variété de *composants logiciels*. Ces briques logicielles préexistantes sont réutilisées pour construire l'application. Produites en continue par des différents constructeurs, elles sont adaptées au contexte précis de leur utilisation. L'Internet devient ainsi un marché libre et dynamique de composants réutilisables.
- Les usages de ces applications utilisent divers *types de terminaux*. Ils apparaissent en continu sur le marché et il est impossible de prévoir à priori l'ensemble des plates-formes cibles en particulier dans le domaine des terminaux mobiles. Nous faisons l'hypothèse que tous possèdent une capacité de calcul et

de la mémoire RAM en quantité suffisante, et divers moyens de connexion (GSM, GPRS, Lan, ou bluetooth).

- Les *besoins des utilisateurs* sont divers et évoluent en continu. Certains de ceux-ci dépendent du contexte physique (position géographique, bruit externe) ou du contexte social (présence d'autres personnes). Par exemple, il peut être nécessaire de compléter un service pour l'adapter aux spécificités de l'utilisateur : déficient visuel ayant besoin d'une IHM adaptée, enfant nécessitant un vocabulaire simple, locuteur français ne sachant lire que le français.

Nous regroupons l'ensemble de ces éléments énumérés ci-dessus dans la notion de *contexte* du service. Ce contexte est variable et il influence en permanence les services selon différents points de vue. L'adaptation de services au contexte est un problème important dont la solution est variable dans le temps.

Les applications à services doivent pouvoir évoluer dynamiquement afin de corriger, améliorer, étendre ou réduire leurs fonctionnalités. La définition abstraite d'une application doit être connue/préservée à l'exécution non seulement pour permettre de guider la composition dynamique de l'application, mais aussi pour pouvoir être modifiée et permettre ainsi l'évolution dynamique de l'application.

Pour ce fait, ces applications à services doivent pouvoir s'adapter dynamiquement aux contextes d'exécution. Les applications doivent de plus en plus être disponibles de façon quasiment ininterrompue. Afin de faire face à la variabilité de leur contexte d'exécution, elles doivent s'adapter en cours d'exécution et en conformité et cohérence avec leur définition abstraite.

2 La notion de contexte

2.1 Définitions du contexte

Il existe de multiples définitions du contexte. Elles sont pour la plupart une énumération de paramètres qui les rendent très abstraites, difficile à exploiter ou au contraire très spécifiques à un domaine. A partir de plusieurs définitions que nous trouvons dans la littérature, nous allons exposer notre propre classification du contexte. Plusieurs tentatives de définition du contexte ont été menées dans plusieurs domaines ces

dernières années et plus particulièrement en informatique ubiquitaire et pervasive avec l'émergence des systèmes sensibles au contexte.

• Le Petit Robert : « *ensemble des circonstances dans lesquelles s'insère un fait* ».

• L'encyclopédie Larousse : « *ensemble des conditions naturelles, sociales, culturelles dans lesquelles se situe un énoncé, un discours* »; ou encore : *« ensemble des circonstances dans lesquelles se produit un événement, se situe une action ».*

• Encyclopédie Larousse *:« ensemble des conditions naturelles, sociales, culturelles dans lesquelles se situe un énoncé, un discours »*. Ou encore : *« ensemble des circonstances dans lesquelles se produit un événement, se situe une action ».*

• Hachette Multimédia :*« ensemble des éléments qui entourent un fait et permettent de le comprendre ».*

• Grand Dictionnaire (Office québécois de la langue française) :*«Énoncé dans lequel figure le terme étudié » ou encore « ensemble d'un texte précédant ou suivant un mot, une phrase, un passage qui éclaire particulièrement la pensée d'un auteur »*. Et, si l'on parle d'informatique : *« ensemble d'informations concernant l'action du stylet, en rapport principalement avec sa localisation à l'écran, qui permet au système d'exploitation de l'ordinateur à stylet de différencier les commandes et l'entrée des données, et de fonctionner en conséquence ».*

(Schilit et al, 1994) fait un parallèle intéressant avec le discours humain :

Quand les humains parlent avec d'autres humains, ils sont capables d'utiliser des informations implicites de la situation, ou contexte, pour augmenter la bande passante de la conversation.

En effet, pour enrichir la conversation et améliorer notre discours, nous y intégrons des informations sur la situation courante à l'instant ou nous parlons. De la même façon, une application pourrait intégrer des informations sur sa situation afin d'enrichir, adapter et améliorer le service fourni. Les personnes à l'origine du terme « sensible au contexte » (context-awareness) sont (Schilit et al, 1994) qu'ils définissent comme « la capacité d'une application et/ou d'un utilisateur mobile de découvrir et réagir aux changements de sa situation ». De très nombreuses autres définitions ont été données (Schilit et al, 1994), (Dey, 2001) mais aucune ne fait pour l'instant autorité.

Depuis quelques années, les avancées sur la mobilité des périphériques et sur la technologie sans –fil ont ouvert de nouvelles perspectives dans divers domaines de

recherche liés à la communication. Cette évolution de l'utilisation de l'informatique a mis en évidence, pour les applications ainsi distribuées, le besoin d'informations supplémentaires à celles habituellement nécessaires aux traitements. Alors que traditionnellement, les applications se contentaient de produire de nouvelles données en sortie à partir de données en entrée, aujourd'hui les traitements peuvent dépendre également de la situation géographique de l'utilisateur, de ses souhaits ou encore des données physiques telles que la température extérieure ou le taux de luminosité. Ces données annexes font partie d'un ensemble appelé données contextuelles ou contexte de l'application. (Cheverst et al, 2002) donnent une définition informelle mais assez représentative :

Le contexte d'exécution d'une application regroupe toutes les entités et situations externes qui influent sur la Qualité de service/Performances (qualitative et quantitative) telle que perçue par les utilisateurs.

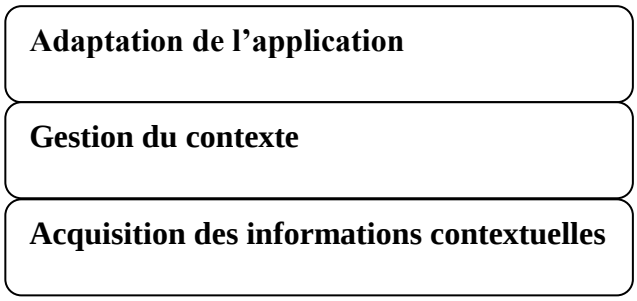

Figure 22: Architecture en couche des applications sensibles au contexte (Demeure et al, 2003).

L'utilisateur n'est pas le seul à percevoir l'influence du contexte sur le service rendu, nous pouvons étendre cette définition au système si nous lui donnons les moyens de prendre conscience de ces influences. Ces définitions montrent qu'à présent les applications doivent intégrer des mécanismes d'acquisition du contexte permettant de le prendre en compte afin de s'adapter par des reconfigurations pour répondre au mieux aux besoins du service rendu.

Cependant le contexte regroupe une multitude de données que nous devons organiser afin d'évaluer comme il se doit les besoins d'adaptation. Le principe architectural de la figure 22 permettant la prise en compte du contexte dans les applications est assez

classique. On en trouvera un exemple dans (Demeure et al, 2003). Il se «résume » à une superposition de couches correspondant à l'acquisition des informations contextuelles, la gestion du contexte puis l'application (et/ou son adaptation).

Ces informations contextuelles proviennent le plus souvent de différentes sources. Tout comme (Chaari et al, 2005) nous distinguons les informations de contexte des informations de l'application. En aucun cas une information contextuelle ne peut être fournie en entrée d'un service de l'application. L'application ne traite pas les informations contextuelles. Ces informations sont recueillies et utilisées par la plate-forme afin d'adapter le comportement et la structure des services. (Demeure et al, 2003)va plus loin en distinguant plusieurs types d'informations contextuelles.

Chacune des informations de contexte a un impact différent sur la QdS (Qualité de service) et donc sur l'adaptation à mener. Nous distinguons les informations reçues de l'utilisateur, de l'environnement, des périphériques et du réseau.

2.1.1 Le contexte d'utilisateur

Ce sont toutes les informations qui concernent ce que l'utilisateur veut pouvoir faire avec l'application. L'utilisateur est très souvent intégré dans le contexte puisqu'il est, la plupart du temps, le seul à percevoir le rendu final de l'application. Une application destinée à un utilisateur doit lui fournir des services. L'utilisateur veut pouvoir émettre des souhaits quant à l'utilisation de l'application. L'application pourra alors être modifiée en conséquence afin de lui fournir un service le mieux adapté possible.

2.1.2 Le contexte d'utilisation

Alors que pour l'utilisateur nous identifions ce qu'il souhaite pouvoir faire avec l'application, nous nous intéressons ici aux spécifications de l'application.

Nous entendons par spécifications tout ce qu'elle doit permettre et ce qu'elle ne doit pas permettre dans son utilisation. Ce sont toutes les situations ou ni l'utilisateur, ni son appareil n'interviennent mais qui demandent tout de même une adaptation comme par exemple quand une alarme se déclenche, quand un niveau sonore est trop élevé, etc.

Il s'agit de modifications de l'environnement qui entrainent, à chaque occurrence, la même adaptation. Chaque fois qu'une situation ainsi décrite est rencontrée, il faudra appliquer une règle bien définie.

2.1.3 Le contexte d'exécution

Dans la plupart des travaux sur la prise en compte du contexte, c'est le contexte d'exécution qui est le plus souvent cité. Comme le dit (Demeure et al, 2003), lorsqu'on parle de contexte d'exécution, il s'agit très souvent d'adapter l'application à l'évolution de l'état des ressources du système sur lequel elle s'exécute. Dans ces ressources nous incluons toutes les propriétés liées au réseau. Tout comme pour des propriétés matérielles comme la mémoire ou l'énergie, nous pouvons mesurer le débit d'un composant ou d'un périphérique et nous pouvons détecter la mobilité.

2.2 Les catégories du contexte

Etant donnée la diversité des informations composant le contexte, il est utile d'essayer de les classer par catégorie pour faciliter leur utilisation. Dans cette section, nous présentons une classification qui synthétise les informations contextuelles utilisées dans les solutions existantes. Les entités principales concernées par la notion de contexte sont des lieux, des personnes ou des objets. Les lieux sont des régions d'espaces géographiques comme des chambres, des bureaux, des bâtiments, des rues ou des zones bien définies. Les personnes peuvent êtres des individus ou des groupes d'individus rassemblés ou répartis. Les objets peuvent êtres des entités physiques, des composants logiciels ou des artefacts (applications, fichiers, ressources…).

Nous classons les informations contextuelles utilisées dans la majorité des travaux existants en quatre catégories principales :

- *Le contexte environnemental* : tel que les personnes ou les objets voisins, la luminosité, la température, le bruit ou le temps, …
- *Le contexte de l'utilisateur* tel que la localisation de l'utilisateur, l'intérêt personnel et les préférences, l'activité, l'émotion, l'handicap, …
- *Le contexte informatique et réseau* tel que l'URL, la bande passante du réseau, la mémoire disponible, le système d'exploitation, …
- *Le contexte de temps* tel que l'historique des actions, l'historique des endroits, l'heure courante et la date.

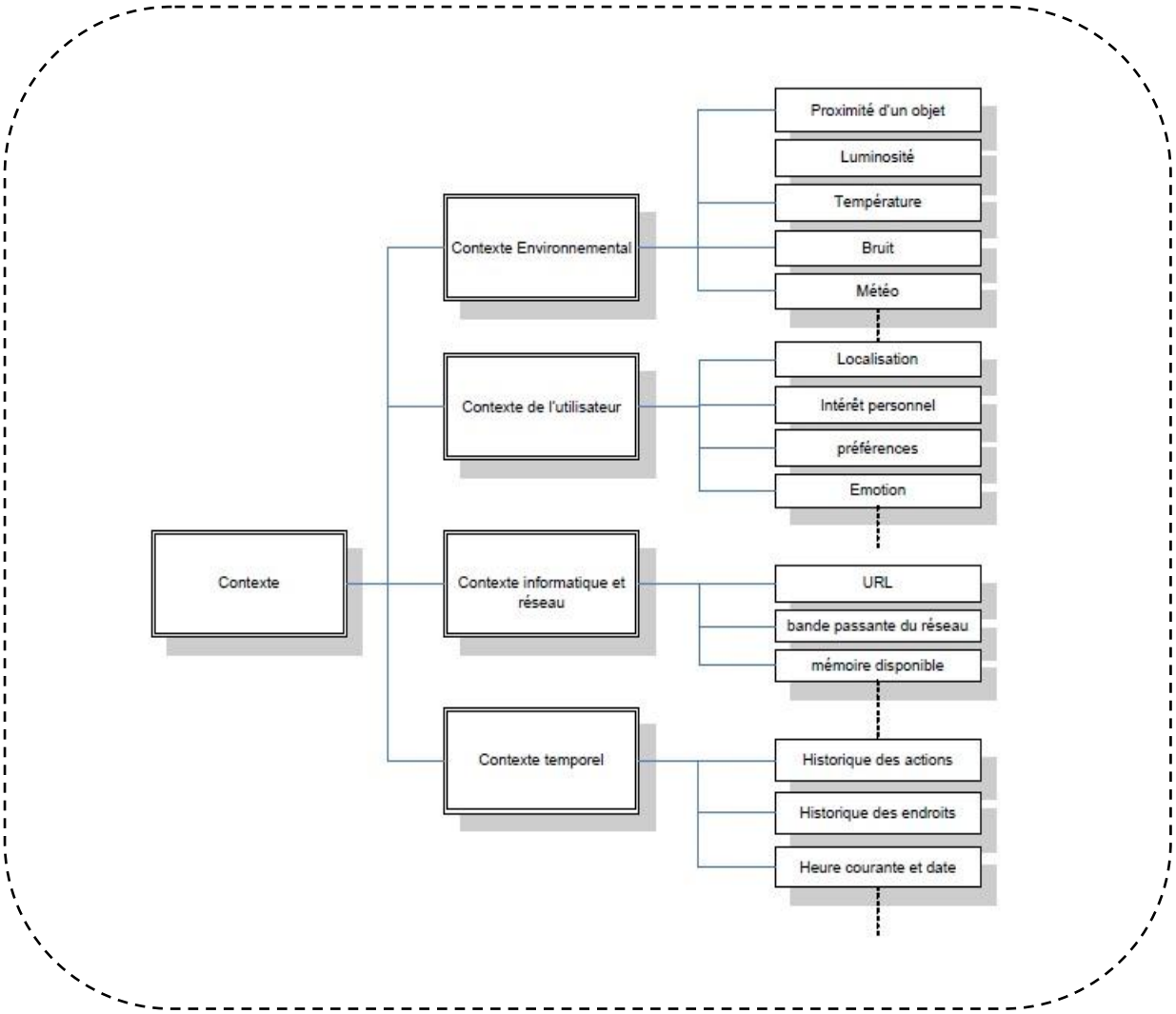

Figure 23: Les différents types d'éléments contextuels

Nous remarquons que ces informations ont des natures très différentes. Certaines sont objectives, facilement mesurables, quantifiables et peuvent être comparées entre elles. D'autres composantes doivent être paramétrées et numérisées comme par exemple les centres d'intérêt d'un individu, sa perception des choses, ses préférences en termes de services ou bien l'émotion qu'il ressent. Certaines informations telles que les préférences devraient être saisies par les utilisateurs eux-mêmes ou bien déduites d'autres informations sur ces derniers. Enfin, certaines informations restent subjectives et difficilement quantifiables, donc difficiles à classifier ou à comparer entre elle.

2.3 La sensibilité au contexte

La sensibilité au contexte vise à prendre en compte le contexte dans les applications. De ce fait, une application sensible au contexte est une application capable de "percevoir" et d'analyser le contexte dans lequel elle s'exécute et de s'y adapter. Ces applications sensibles au contexte adaptent leur comportement au contexte de l'utilisateur afin de mieux le satisfaire. Les travaux de (Chaari et al, 2006) sont les premiers à avoir introduit la notion de la sensibilité au contexte et l'ont défini comme étant la capacité d'une application à découvrir e tà réagir aux modifications de l'environnement où se trouve l'utilisateur. Les systèmes sensibles au contexte sont définis comme des systèmes qui s'adaptent à la localisation de l'utilisateur et à l'ensemble des personnes et des dispositifs proches ou accessibles, ainsi qu'aux changements dans le temps de ces éléments (Abbas et al, 2007).

Plusieurs définitions de cette notion sont données dans la littérature :

(Akkawi et al, 2007), par exemple, définissent les applications sensibles au contexte comme des applications dont le comportement peut varier en fonction du contexte de l'utilisateur.

Selon (Mc Kinley et al, 2004) un système sensible au contexte adapte les informations ou les services qu'il offre aux circonstances courantes d'utilisation.

(Cetina et al, 2009) considèrent que les systèmes sensibles au contexte sont des systèmes qui peuvent adapter la présentation des informations afin de les rendre plus pertinentes au contexte d'un utilisateur en tenant compte, par exemple, des intérêts signalés par l'utilisateur ou de son contexte environnemental.

Selon (Alferez et al, 2011a) les systèmes « sensibles au contexte » ou « adaptatifs au contexte » sont des systèmes qui peuvent découvrir et utiliser des informations contextuelles telles que la localisation de l'utilisateur, les caractéristiques de ses dispositifs, etc.

(Koning et al, 2009) définissent la sensibilité au contexte comme la capacité d'un système à percevoir la situation dans laquelle se trouve l'utilisateur et d'adapter en conséquence le comportement du système en termes de services, de données et d'interfaces.

Selon (Cardoso et al, 2002), les systèmes sensibles au contexte sont des systèmes capables de retourner à l'utilisateur des informations pertinentes et adaptées à ses

besoins et son contexte qui influence son comportement lors de son interaction avec les systèmes d'information.

Plus généralement, nous pouvons dire qu'un système sensible au contexte est un système dont le comportement ou la structure peut varier en fonction de l'état de l'espace des informations de contexte. Un tel système peut être considéré comme un système qui est capable de détecter, à un instant donné, un ou plusieurs éléments du contexte dans lequel se trouve l'utilisateur et de fournir, en retour, soit des informations, soit des services adaptés à ce contexte et aux changements d'un ou plusieurs éléments du contexte.

3 Etude de l'existant

Plusieurs travaux d'adaptation dynamique sont basés sur les modèles à base de composants, un modèle de composant émerge dans le monde industriel : les services Web.

Fondamentalement, un service Web est un composant logiciel spécial qui est recherché, lié, et exécutés à l'exécution et permet aux systèmes d'interagir par le biais des protocoles Internet standard (Baresi et al, 2011). Afin d'atteindre le plein potentiel des services Web, ils peuvent être combinés pour obtenir des fonctionnalités spécifiques. Si la mise en œuvre d'une logique- métier de service Web implique l'invocation d'autres services Web, il est appelé un service composite. Dans le cas de services Web, l'assemblage a une dimension temporelle et est réalisé sous la forme d'un ***Workflow.***

Les travaux (Narenda et al, 2007), (Sonntag et al, 2011), (Moser et al, 2008), (Fleurey et al, 2009) ont ciblé l'évaluation des attributs qualité des orchestrations de services en se basant sur des règles d'agrégation. Ces travaux partagent le même principe. L'idée de base consiste à définir des règles d'agrégation pour chaque patron de workflow et pour chaque attribut qualité. Dans la plupart des cas, les règles sont définies pour un couple de patrons de workflow (dénotés souvent comme « patrons de composition » (Sonntag et al, 2011), (Moser et al, 2008)) sauf pour les patrons « séquence » et « boucle » qui sont considérés individuellement.

Les travaux connexes sur l'adaptation dynamique des compositions de services peuvent être classés en trois groupes. Le premier groupe soutient l'adaptation dynamique au niveau de la langue (Erradi et al,2005), (Gadellini et al, 2010), Mosincat et al, 2008),

(Oliveira et al, 2014), (Canal et al, 2006). Cette approche peut nuire au raisonnement des adaptations avec des scripts complexes et peut être sujettes à l'erreur (Benmokhtar et al, 2005). Le deuxième groupe se concentre sur les mécanismes de mise en œuvre de bas niveau pour l'auto-adaptation (Benmokhtar et al, 2005), (Cheung-Foo-Wo, 2006a), ((Blay-Fornarino et al, 2004), (Berger, 2011). Cette approche manque de soutien pour l'analyse de la variabilité inhérente à l'adaptation dynamique au moment de la conception, et peut nuire à raisonner sur des adaptations avec des scripts complexes et sujettes à l'erreur.

Le troisième groupe, procède par l'utilisation du système de transition de reconfiguration (RTS), dans (Baroudi et al, 2017) les auteurs définissent un modèle de graphe de transition au moment de la conception. Les nœuds du graphe représentent les configurations du système, et les arcs (arrêtes) représentent les transitions de reconfiguration à savoir, les opérations pour adapter un système d'une configuration à une autre. Cependant, le nombre de configurations valides peut croitre de façon exponentielle pour un grand nombre de points de variabilité du système dû à la combinaison de caractéristiques. Par conséquent, la construction d'un tel modèle de diagramme peut être impossible dans de nombreux domaines.

4 Objectif :

Notre objectif est de proposer une architecture logicielle qui rend possible l'adaptation dynamique de services construits par assemblage de composants, en fonction de contextes d'utilisation variés. Dans le cadre de cette première expérimentation le contexte concerne les besoins des utilisateurs. La particularité principale de notre proposition est que le comportement de chaque service par rapport à son contexte d'utilisation est évalué à partir de l'analyse du comportement de chaque composant constituant le service. Pour cela, nous utilisons des profils qui décrivent non seulement les éléments du contexte mais aussi chaque composant constituant le service. Un Adaptateur analyse la conformité entre les différents profils de chaque composant et les profils des éléments du contexte. L'Adaptateur détecte les points d'inadaptation, cherche et applique aux différents composants du service les modifications nécessaires pour rétablir cette compatibilité par modification de paramètres de configuration, par ajout, retrait ou remplacement de composants.

5 Architecture pour l'adaptation dynamique de service :

L'architecture proposée est constituée de trois parties, illustrées dans la figure 24 :

- *Partie modifiable*. Cette partie est construite principalement par le service, constitué lui-même d'un assemblage de composants. Les éléments modifiables sont les composants, ainsi que les différentes interconnexions entre eux.
- *Partie monitoring*. Cette partie est représentée par des moniteurs qui observent les ressources et les profils utilisateurs. Ces éléments fournissent les données nécessaires à une description complète du service appelée méta-description de l'ensemble service-contexte.
- *Partie contrôle*. Cette partie est représentée par : l'Adaptateur qui, à partir de la description de l'ensemble service-contexte, décide la modification nécessaire pour adapter le service ; et d'un Assembleur qui exécute ce que l'Adaptateur lui dicte. L'Adaptateur utilise des composants existants trouvés dans la base de composants comme des solutions d'adaptation.

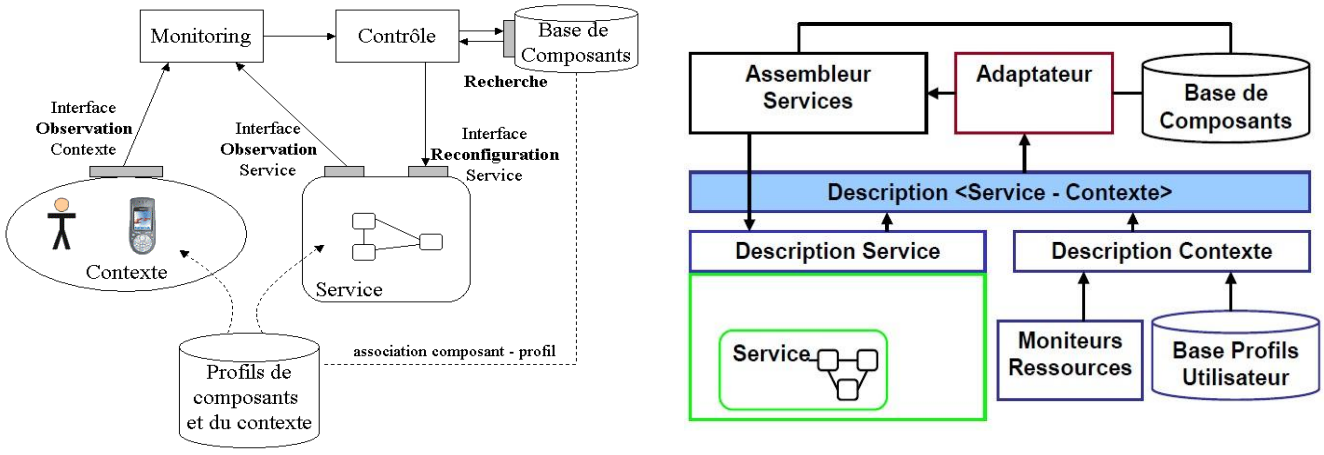

Figure 24: Architecture pour l'adaptation dynamique de service

5.1 Exemple illustratif :

Un service de forum électronique permet à une communauté d'étudiants d'échanger des renseignements concernant les activités scientifiques et culturelles se passant au sein de l'université.

Pour des raisons d'usage, la langue de discussion est le français. Les étudiants membres ont la possibilité d'écrire et de lire sur le forum.

5.2 La méta-description de l'ensemble service-contexte :

La figure 24 présente une perspective sur la méta-description de l'ensemble service-contexte correspondante au scénario présenté dans l'exemple illustratif. Nous trouvons trois plans différents:

- Le plan utilisateur (PU) contenant les éléments physiques du contexte.
- Le plan des composants contenant l'ADL (Architecture Description Language) du service et les IDL (Interface Definition Language) de composants.

Les ADLs décrivent les architectures logicielles. Les règles de réécriture capturent et regroupent les adaptations sous forme d'ensembles de règles (schémas) et facilitent leur sélection.

- Le plan de profils qui réunit le profil du service et le profil du contexte et qui indique le fonctionnement du service.

5.2.1 Le plan utilisateur – éléments physiques du contexte

Le plan physique contient les éléments physiques du contexte. Dans ce cas il s'agit des utilisateurs du service de forum. Les éléments trouvés dans ce plan ont des projections dans le plan de profils. Ces projections forment la méta-description du contexte.

5.2.2 Le plan de composants – service, composants, assemblage

Le plan de composants contient le service S sous forme d'assemblage de composants logiciels. Pour notre exemple il est constitué du :

- Composant A : Afficher les messages publiés sur le forum,
- Composant E : Ecrire un nouveau message
- Composant F : Forum qui contient tous les messages publiés sur le forum.

L'IHM du service S est construite à partir d'une composition des IHM des composants E et A. Ce plan de composants représente ainsi la partie syntaxique de la méta-description du service. Pour décrire le service dans ce plan nous utilisons un langage de type ADL qui décrit l'architecture interne du service, des descripteurs IDL et des interconnections qui sont des fichiers MANIFEST.MF pour les composants. Nous avons inclus aussi l'IHM parmi les interconnections (ou bien les connecteurs).

5.2.3 Le plan de profils – profil utilisateur, profil de composants, composition de profils

Le plan des profils est essentiel pour l'adaptation car il représente la partie sémantique de la méta-description de l'ensemble service-contexte. Pour l'exemple choisi il contient :

Le profil du service PS, et le profil utilisateur Pu. PS est le résultat de la composition de profils des composants qui constituent le service S : Pe, Pa, Pf. Ce plan contient la sémantique de l'ensemble service-contexte.

Le profil utilisateur contient, dans notre exemple, un seul paramètre : langue = 'Ar' indiquant un utilisateur écrivant en arabe. Le profil d'un composant indique comment ce dernier fonctionne par rapport aux paramètres du contexte. Voici le profil pour un composant de traduction arabe - français :

```
<profile>
        <component>Translation AR_FR</component>
            <point>
                <interface>Translation</interface>
                <method>translate</method>
                <argument>text</argument>
                <argtype>String</argtype>
                <precondition>langue = 'AR' </precondition>
            </point>
            <point>
                <interface>Translation</interface>
                <method>translate</method>
```

<returntype>String</returntype>

<function>=</function>

<postcondition>**langue** = ‘FR’ </postcondition>

</point>

</profile>

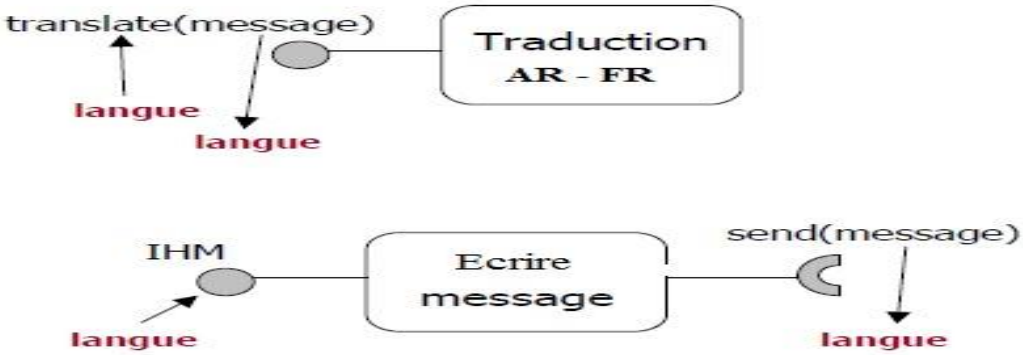

Figure 25: Exemple de composant de traduction

Ce profil indique le comportement du composant par rapport à la langue : au niveau de l’interface « Translation » du composant « TranslationAR_FR » existe une pré-condition sur le paramètre « langue » qui demande la valeur « AR ».

La valeur retournée aura la valeur « FR ». Ce composant modifie donc la langue et c’est grâce au profil que l’Adaptateur peut découvrir ce fait.

Le profil du service, qui est un composant composite, résulte par la composition des profils de composants E, A, F : F impose langue = ‘FR’ par convention, F est connecté avec E mais E est neutre de point de vue langue ainsi que la condition se propage au niveau de E. L’IHM de E est connecté à l’IHM de S ainsi que la condition se propage au niveau de l’IHM de S.

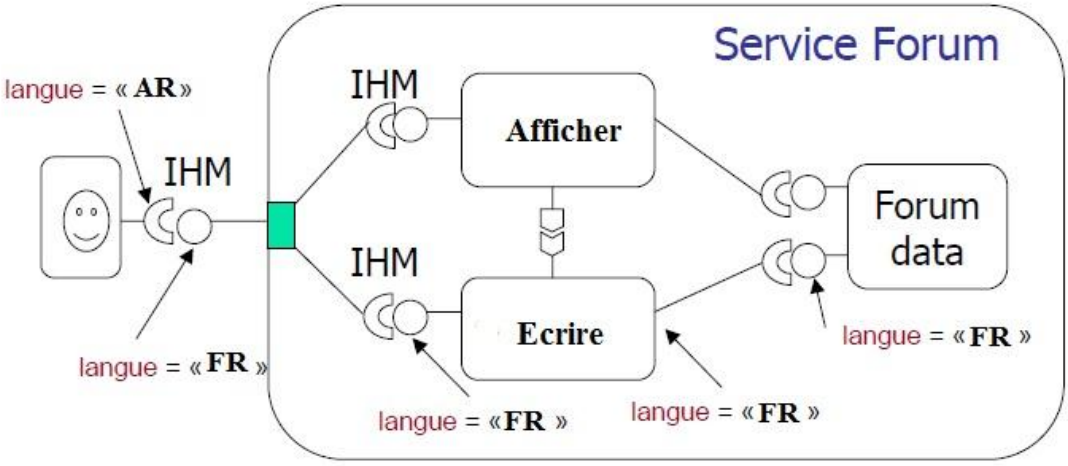

Figure 26: Composition de profils

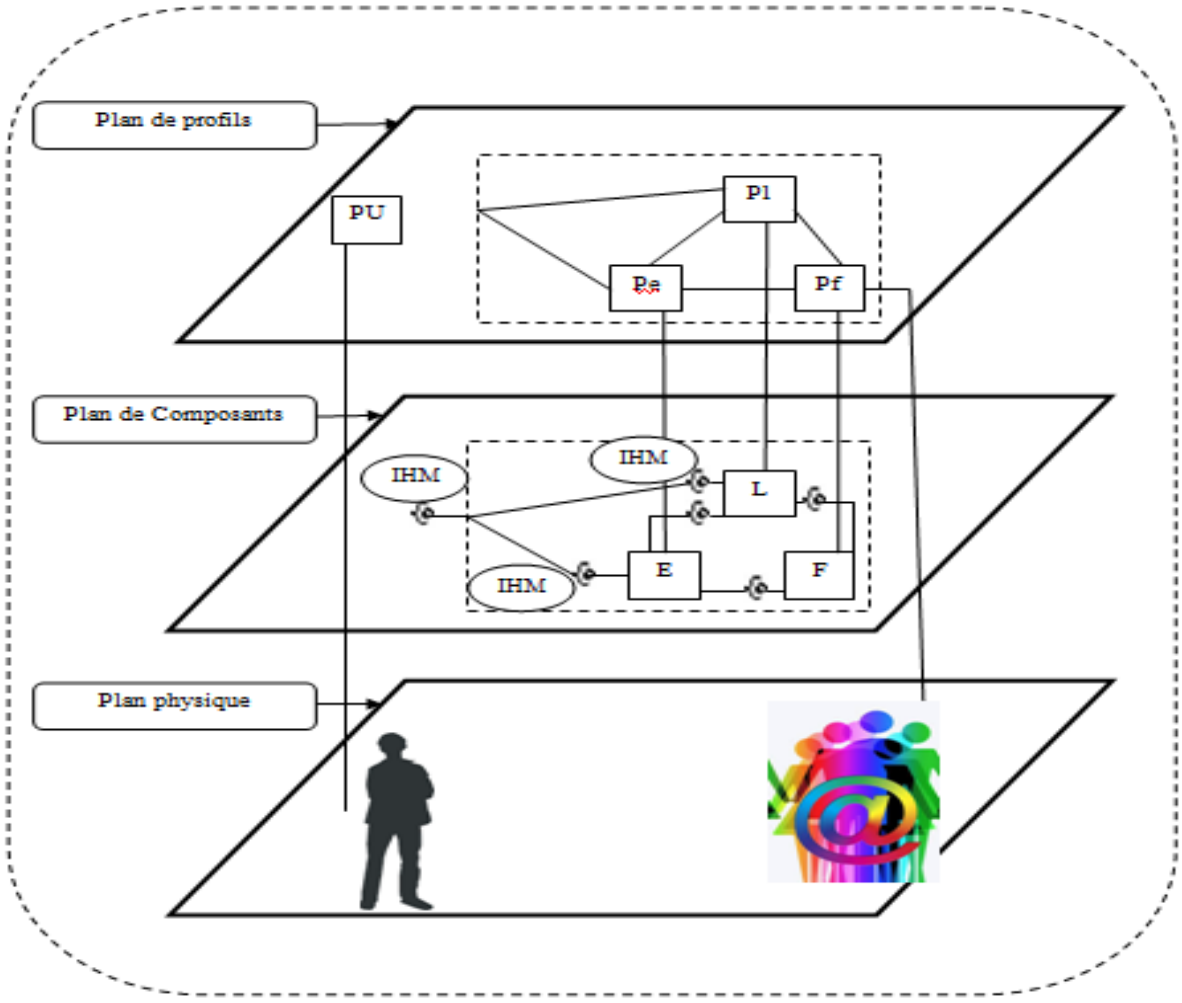

Figure 27: La méta-description de l'ensemble service-contexte « les différents plans de vue »

5.2.4 Les axiomes d'adaptation de l'ensemble service-contexte

Pour la partie sémantique de l'ensemble service-contexte, celle qui correspond au plan de profils, nous avons besoin de définir les axiomes (évidences ou conditions) qui sont vérifiés si un service est adapté à son contexte. Pour cet exemple il existe un seul axiome : nous avons besoin d'une adaptation de l'ensemble service-contexte si les valeurs des paramètres des profils sont contradictoires. Dans notre exemple, si l'utilisateur écrit en Arabe, il existe une contradiction pour le paramètre « langue », (Ar <> Fr) au niveau de l'IHM du service S. Dans ce cas l'axiome d'adaptation n'est pas vérifié.

5.3 L'Adaptateur – algorithme

L'Adaptateur doit pouvoir adapter le service, son automatisation passe par l'utilisation de formalismes dynamiques et procède en trois étapes (Canal et al, 2006) :

- Une technique de détection d'incompatibilités est développée.
- Une description abstraite des propriétés du système adapté, du médiateur entre composants (l'adaptateur) ou de simples correspondances entre interfaces est fournie : *le Mapping d'adaptation.*
- Le processus d'adaptation est produit ensuite, automatiquement à l'aide de ce *Mapping* et des interfaces des composants à adapter. L'adaptateur permet, lorsqu'il est placé au sein du contexte d'exécution, d'assurer la correction de système.

Le *Mapping* d'adaptation peut être obtenu automatiquement grâce à des techniques du domaine des services Web sémantiques (Ben Mokhtar et al, 2005), comme il peut être fourni par l'architecte du logiciel ce qui est le cas que nous traitons.

Nous proposons un adaptateur qu'il doit effectuer les opérations suivantes :

– *Vérification*. Pour chaque paramètre des profils les axiomes d'adaptation sont vérifiés, s'ils sont tous respectés nous avons un service adapté.

– *Recherche d'une solution d'adaptation.* Si pour au moins un paramètre, un axiome n'est pas respecté, il faut rechercher une adaptation :

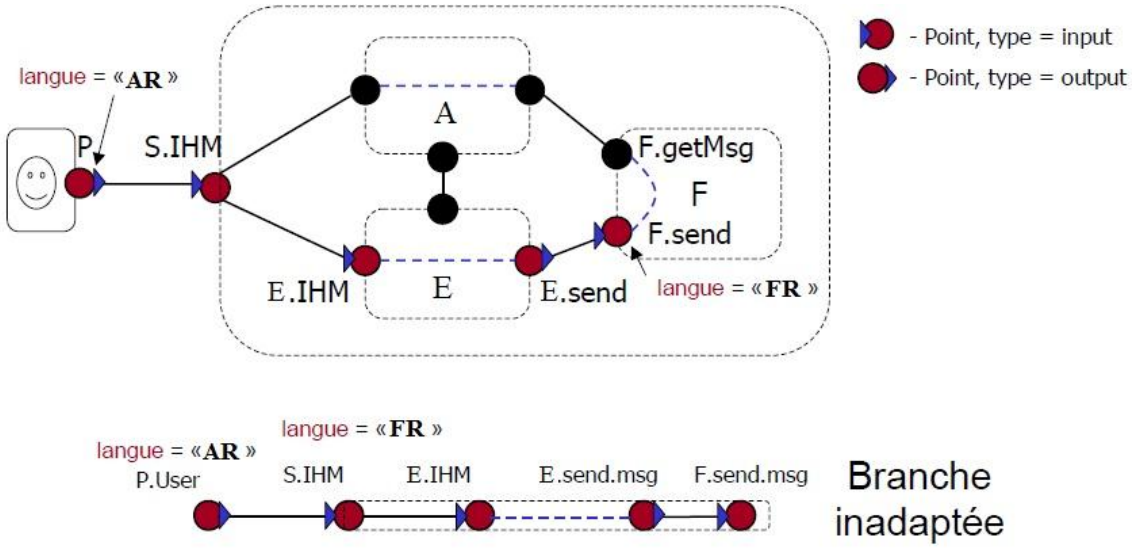

Figure 28: Graphe de propagation du paramètre langue.

− Pour chaque paramètre qui ne vérifie pas les axiomes, l'Adaptateur construit un graphe qui a comme nœuds les interfaces de composants qui ont une relation avec ce paramètre et dont les arcs sont les interconnexions.

− Dans ce graphe l'adaptateur trouve les branches (enchaînement des nœuds) au niveau desquelles se trouvent des inégalités de valeur. Pour chaque branche l'Adaptateur cherche des composants, l'insertion de ces derniers dans la branche, permet de rétablir l'égalité des valeurs.

─ *Application de la solution*. Si l'adaptateur est arrivé à ce point, il peut appliquer la solution trouvée.

6 Prototype :

Nous proposons dans le prototype (figure 29), que le service est constitué par les composants :

A – Afficher forum,

E – Ecrire un message,

Proxy F – un connecteur local vers le composant distant F – Forum.

Le composant T – Traduction, est ajouté dynamiquement à la demande de l'Adaptateur, en utilisant :

- ✓ La plateforme Wcomp (Cheung-Foo-Wo, 2006a) : qui offre un environnement de développement à base de composants logiciels, elle permet la création dynamique des interconnexions avec ISL4Wcomp. L'adaptation dynamique sur Wcomp se fait par l'ajout, le retrait, la connexion et la déconnexion des composants logiciels pendant l'exécution de l'application.
- ✓ Le langage d'interactions ISL4Wcomp (Blay-Fornarino, 2004) (Interaction Specification Language For Wcomp): il se base sur le langage de spécification d'interactions nommé ISL (*Interaction Specification Language)*,qu'il permet de décrire des schémas d'interactions entre les objets (Berger, 2011). ISL4Wcomp quant à lui adapte ces spécifications pour la prise en compte d'interactions basées sur des messages ou événements dans des composants dans les aspects d'assemblage.

L'Adaptateur trouve ce composant dans l'annuaire de composants, la recherche s'effectue à partir du profil du composant en respectant le typage des connexions.

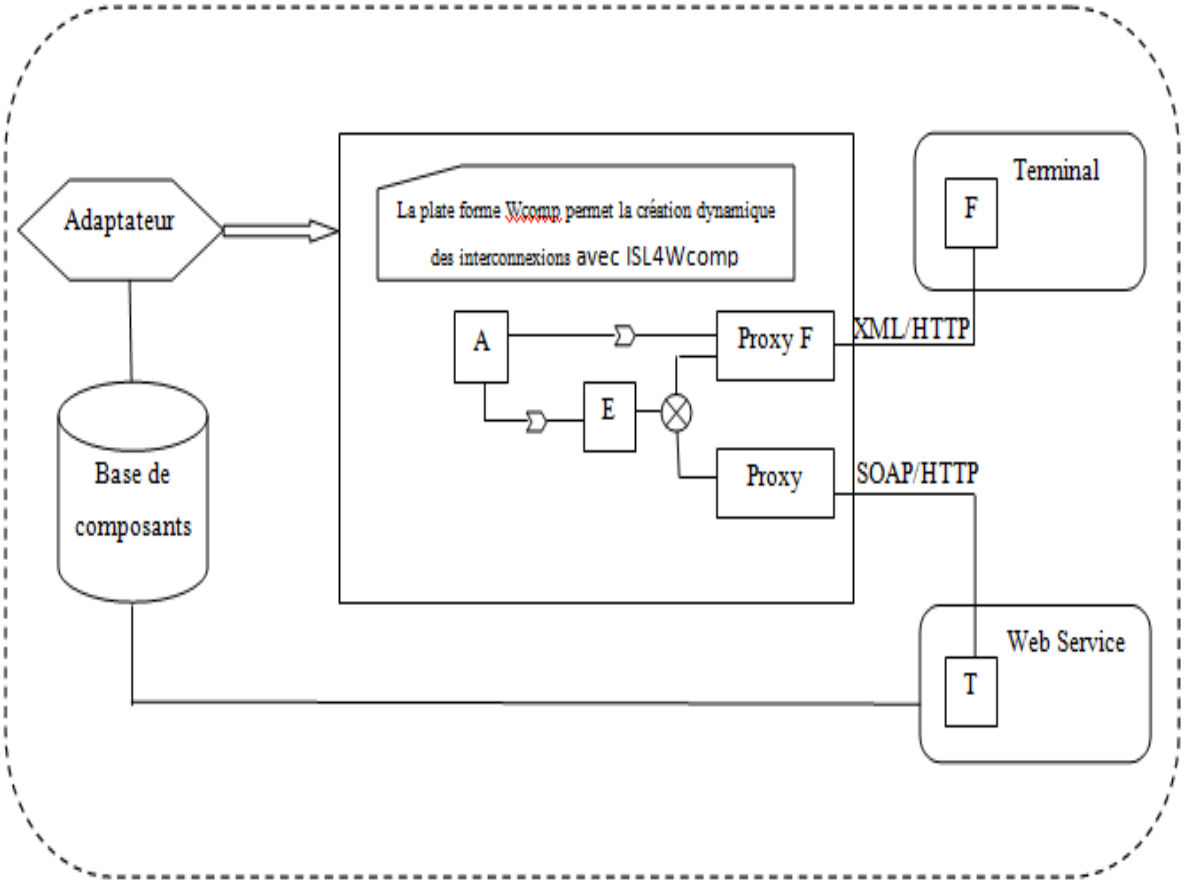

Figure 29: Schéma du prototype

7 Conclusion

Dans ce chapitre nous avons présenté une architecture qui permet l'adaptation dynamique de services. Notre proposition est basée sur une méta-description de *l'ensemble service-contexte* dans lequel nous avons un ensemble limité des axiomes d'adaptation basés sur la sémantique du service. Il n'est pas nécessaire de décrire les différentes règles d'évolution, celles-ci seront découvertes par analyse des cas d'inadaptation sous la seule réserve que chaque composant soit décrit avec son profil.

Pour prouver le fonctionnement de cette architecture nous avons implémenté un prototype de service de forum. Ce service est crée initialement pour des utilisateurs francophones mais l'architecture proposée permet d'adapter ce service par ajout dynamique d'un composant de traduction si la langue de l'utilisateur n'est pas le français.

Le principal désavantage de notre l'architecture est sa complexité. Il reste encore à généraliser et simplifier le modèle afin que les différents paramètres de profils se composent plus simplement tout en gardant une richesse nécessaire à l'expression des axiomes d'adaptation.

Malgré ces limitations et difficultés, nous considérons que l'avenir de l'adaptation appartient à la composition sémantique des services. Pour réaliser cela, le fonctionnement de l'ensemble service-contexte doit être compréhensible aussi pour la machine et pas seulement pour le constructeur humain de services.

Conclusion Générale

Le travail présenté dans ce mémoire s'inscrit dans le cadre du développement à base de services. Nous avons abordé le domaine d'adaptation dynamique des applications à base de services, l'objectif était de donner une vue générale sur ce domaine en introduisant les problèmes à résoudre, les contraintes à respecter et défis à surmonter pour rendre l'adaptation possible. L'approche à services fournit ainsi deux propriétés importantes : le faible couplage entre clients et fournisseurs de services, et la liaison tardive. Grâce à ces propriétés, l'approche à services permet la construction d'applications flexibles et dynamiques par composition de services.

De son côté, l'adaptation tient compte essentiellement des évolutions de l'utilisateur et de son environnement et non des conditions d'exécution des services eux-mêmes.

Afin d'aborder plus particulièrement cette problématique d'adaptation, il est nécessaire de considérer la notion de contexte.

La variabilité des contextes dans lesquels s'exécutent les applications rend difficile, voire impossible, d'avoir une connaissance complète, lors de la conception et du développement des applications, des conditions précises dans lesquelles les applications seront utilisées : les services et les dispositifs qui seront disponibles à l'exécution ou ceux qui seraient mieux adaptés à un moment donné. Même si les décisions prises à la conception d'une application sont exactes au début de son exécution, l'évolution du contexte peut requérir son adaptation dynamique afin de lui permettre de continuer à fonctionner ou de tirer parti des services et des dispositifs apparus en cours d'exécution.

Aussi, le dynamisme soulève de nombreux besoins et défis pour la construction d'applications à services comme par exemple :

- ✓ La définition d'une application par l'ensemble des propriétés et des contraintes à satisfaire lors de sa composition.
- ✓ La composition incrémentale et dynamique de l'application,
- ✓ La vérification de la cohérence et de la conformité de la composition vis-à-vis de sa définition,
- ✓ L'adaptation dynamique de la composition face à la variabilité du contexte d'exécution,
- ✓ L'évolution de la définition d'une application.

Afin de répondre à ces besoins et défis, nous avons proposé un prototype d'une architecture permettant l'adaptation dynamique de service. Notre proposition à pour but de constituer un service par un assemblage de composants, elle est basée sur une méta-description de *l'ensemble service-contexte* dans lequel nous avons un ensemble limité des axiomes d'adaptation basés sur la sémantique du service. Il n'est pas nécessaire de décrire les différentes règles d'évolution, celles-ci seront découvertes par analyse des cas d'inadaptation sous la seule réserve que chaque composant soit décrit avec son profil.

Pour prouver le fonctionnement de cette architecture nous avons implémenté un prototype de service de forum. Ce service est crée initialement pour des utilisateurs francophones mais l'architecture proposée permet d'adapter ce service par ajout dynamique d'un composant de traduction si la langue de l'utilisateur n'est pas le français.

Le travail que nous avons réalisé dans cette thèse a permis, d'une part d'explorer différents travaux portant sur les applications à base de services et leur adaptation dynamique, et d'autre part, de contribuer au développement d'une architecture d'adaptation de services.

L'évaluation de notre l'architecture réside en sa complexité, ce qui nous amène à prévoir de simplifier le modèle afin que les différents paramètres de profils se composent plus simplement tout en gardant une richesse nécessaire à l'expression des axiomes d'adaptation.

Perspectives

Pour notre travail, nous nous sommes focalisés sur la notion de contexte qui concerne seulement le profil de l'utilisateur qui ne peut lire que dans la langue « Française », nous pouvons le généraliser pour plusieurs langues.

La diversité des terminaux employés par l'utilisateur pour accéder aux ressources (PC, PDA, téléphone mobile,..etc.) en utilisant plusieurs types de réseaux (local, wifi,…etc.) génère un besoin croissant d'adapter dynamiquement des services au contexte de l'utilisateur.

Tester plus de scénarios pour l'adaptation dynamique. Par exemple multiplier les composants de traduction et en choisir celui qui offre la meilleur la qualité de service.

Proposer un modèle générique pour l'architecture d'adaptation dynamique de services.

Le principal désavantage de notre l'architecture est sa complexité. Il reste encore à généraliser et simplifier le modèle afin que les différents paramètres de profils se composent plus simplement tout en gardant une richesse nécessaire à l'expression des conditions d'adaptation.

Malgré ces limitations et difficultés, nous considérons que l'avenir de l'adaptation appartient à la composition sémantique des services. Pour réaliser cela,le fonctionnement de l'ensemble service-contexte doit être compréhensible aussi pour la machine et pas seulement pour le constructeur humain de services.

Bibliographie

Abbas.K, Verdier.C, Flory.A : «Exploiting profile modeling for web-based information system». In proceedings of the international workshop on personalized access to web information (PAWI'07), pp 591-596, Nancy, France, 2007.

AkanO.B, Akyildiz.I.F : « ATL : An adaptative transport layer suite for next generation wirless internet ». IEEE journal on selected areas in communications, June 2004, Vol 22, N°5, pp 802-817.

Akkawi.F, Akkawi.K, Bader.A, Ayyash.M, Fletcher.D, Alzoubi.K: «Software adaptation: A conscious design for oblivious programmers». In proceedings of the IEEE aerospace conference, pp 1-12, March 2007.

Alferez.G.H, Pelechano.P.V: «Context-aware autonomous web services in software product lines». In proceedings of the 2011, 15th international software product line conference. SPLC'11, IEEE computer science, Washington, DC, USA, V 2011a, pp 100-109.

Allen.R.J : « A Formal Approach to software architecture ». PhD thesis, CMU-CS- 97-144, 1997. School of computer science Carnegie Mellon University Pittsburgh.

André.P, Ardourel.G,Attiogbé.C, Habrias.H, Stoquer.C : « Vérification de conformité des interactions entre composants ». OCM 2005, Berne -Suisse.

Andrews.T, Curbear.F, Dholakia.H, Goland.Y, Klein.J, Leymann.F, Liu.K, Roller.D,Smith.D, Thatte.S, Trickovic.I, Weerawarana.S : « Business process execution language for web services ». Version 1.1 IBM specification, 2003.

Arentze.A.T, Hofman.F, Timmermans.J.P : « Deriving rules from activity diary data : a learning algorithm and results of computer experiments ». Journal of geographical systems, Vol 3, N°4, pp 325-346, 2001.

Arkin.A, Askary.S, Fordin.S, Jekeli.W, Kawaguchi.K, Orchard.D, Pogliani.S, Riemer.K, S.Struble.S, Takasci.P, Trickvic.I, Zimek.S : « Web service choreography interface ». WSCI 1.0 W3C Note, 2002.

Arsanjani.A : « Service-oriented modeling and architecture », November 2004.

Arenaza.N : « Composition semi-automatique des web services ». Projet de master, Ecole polytechnique fédérale de Lausane, Février 2006.

Baresi.L, Guinea.S: «Self-supervising BPEL processes». IEEE transactions on software engeneering, Vol 37, pp 247-263, March 2011.

Baroudi.M.Y, Bendimerad.F.T : « Service oriented approach state of the art ». Journal of CTA, Vol 4, November 2013.

Baroudi.M.Y, Benamar.A, Bendimerad.F.T : «Dynamic service adaptation architecture». Journal of advanced computer science and application, Vol 5, N°3, pp 30-35, May 2017.

Barros.A, Dumas.M, Oaks.P : « Standards for web service choreography and orchestration : status and perspectives ». In the 1st international workshop on web service choreography and orchestration for business process management, Nancy, France, September 2005, pp 1-15.

Benatallah.B, Dumas.M, Fauvet.M-C, Rabhi.F.A, Sheng.Q.Z : « Overview of some patterns for architecting and managing composite web services ». ACM Sigecom Exchanges, 2002, Vol 3, N°3, pp 9-16.

Benatallah.B, Dijkman.R, Dumas.M, Maamar.Z : « Service composition : concepts techniques, tools and trends ». In Z.Stojanovic, A.Dahanayake, Editions. Service oriented software engineering : challenges and practices. Idea Group Inc (IGI), 2005, pp 48-66.

Benazzouz.Y, Beaune.P, Ramaparany.F, Boisier.O : « Modeling and storage of context data for service adaptation ». In Schahram Dustdar (Eds) Quan Z.Sheng, Y.Jian, enabling context-aware web services : methods, architectures, and technologies, pp 470-490, CDC press Taylor and Francis Group, 2010.

Ben Mokhtar.S, Georgantas.N, Issarny.V: «Adhoc composition of user tasks in pervasive computing environments». Software composition, Springer-Verlag, LNCS 3628, pp 31-46, 2005.

Bhusate.A, Pitt.J: «Pervasive adaptation for enhancing quality of experience». In Proceedings of the 2nd perada workshop on pervasive adaptation, pp 3-8, Edinburg, Scotland, 2009.

Berger.L: «Mise en œuvre des interactions en environnements distribués, compiles et fortement types: le modèle MICADO». Thèse de doctorat, université de Nice-Sophia antipolis, Octobre2011.

Beth.M : « Retail user assistant : evaluation of a user-adapted performance support system ». In proceedings of the 4th international conference on human-computer interaction, pp 45-55, London, UK, 1994, Springer-Verlag.

Booch.G, Object-oriented analysis and design with applications (2nd edition). Object-oriented software engineering. Addison-Wesley Professional, 1993.

Blay-Fornarino.M, Charfi.A, Amsellem.D, Pinna-Dery.A-M, Riveill.M: «Software interactions». Journal of object technology, Vol 3, N°10, pp 161-180, 2004.

Brdiczka.O, Crowley.J.L, Reignier.P : « Learning situation models for providing context-aware services ». In proceedings of the 4th international conference on universal access in human-computer interaction, Vol 4555 of lecture Notes in computer science, pp 23-32.

Brown.P.J, Bovey.J.D, Chen.X: «Context-aware applications: from the laboratory to the marketplace». In IEEE personal communications, Vol 4, N°5,pp 58-64, 1997.

Bridges.P.G, Chen.W.K, Hiltunen.M.A : « Supporting coordinated adaptation in networked wireless systems.8th workshop on Hot Topics in Operating Systems, May 2001, Elmau, Germany, pp 162-168.

Bruneton.E, Coupaye.T, Stefani.E.J : »The fractal component model ». The Object Web Consortium Specification, Draft 5, 2004, version 2.03.

Canal.C, Murillo.J.M, Poizat.P: «Software adaptation». Special issue on coordination and adaptation techniques, Vol 12, N°1, pp 9-31, 2006.

Capra.L, Emmerich.W, Mascolo.C : « Carisma : Context-aware reflective middleware system for mobile applications. IEEE transactions on software engineering, Vol 29, pp 929-945, 2003.

Cardoso.J, Miller.J, Sheth.A, Arnold.J: «Modeling quality of service for workflows and web service processes». Journal of web semantics, Vol 1, pp 281-308, 2002.

Cassagnes.C, Roose.P, Dalmau.M : « Kalimucho : Software architecture for limited mobile devices ». ACM Sigbed review, Vol 6, N°3, pp 1-6, 2009.

Cervantes.H ,« Vers un modèle à composants orienté services pour supporter la disponibilité dynamique ». Thèse de doctorat de l'Université Joseph Fourier de Grenoble, Mars 2004.

Cervantes.H, Hall.R.S : « Autonomous adaptation to dynamic availability using a service-oriented component model ». In 26th International conference on software engineering (ICSE'04), pp 614-623, 2004.

Cetina.C, Giner.P, Fons.J, Pelechano.V: «Autonomic computing through reuse of variability models at runtime: the case of smart homes». Computer Vol 42, pp 37-43, October 2009.

Chaari.T : « Adaptation d'applications pervasives dans des environnements multi contextes ». Thèse de doctorat de l'Université de Lyon, France, 28 Septembre 2007.

Chaari.T, Laforest.F : « Génération et adaptation automatiques des interfaces utilisateurs pour des environnements multi-terminaux ». Projet Sefagi : simple environnement for adaptable graphical interface. Revue ingénierie des systèmes d'information, N° spécial systèmes d'information pervasifs. Vol 9, N°2, pp 11-38, 2004.

Chaari.T, Zouari.M, Laforest.F : « Ontology based context-aware adaptation approach ». In book of context-aware mobile and ubiquitous computing for enhanced usability : adaptive technologies and applications, pp 26-58, Hershey-New York, 2009. Information science reference.

Chaari.T, Dejene.E, Laforest .F, Scuturici.V-M: «A comprehensive approach to model and use context for adapting applications in pervasive environments». Journal of system software, Vol 80, N°12, pp 1973-1992, 2007.

Chaari.T, Ejigu.D, Laforest.F, Stururici.V-M : «Modeling and using context in adapting applications to pervasive environments». Pervasive services, ACS/IEEE international conference on pp 11-120, 2006.

Chaari.T, Laforest.F, Flory.A : «Adaptation des applications au contexte en utilisant les services web». In UbiMob'05 : la 2ème journée francophone mobilité et ubiquité, pp 111-118, Grenoble, France, 2005.

Charif.Y, Sabouret.N : « Coordination in introspective multi-agent systems », proceedings of the international conference on intelligent agent technology (IAT'07), pp 412-415,Sillicon valley, California, USA, 2007.

Chassot.C, Dira.K, Guennoun.K : « Towards autonomous management of QoS through model-driven adaptability in communication-centric systems ». International transactions on systems science and applications, Vol 2, N°3, pp 255-264, 2006.

Chen.W.K, Hiltunen.A.M, Schlichting.R.D : « Constructing adaptive software in distributed systems ». In ICDCS'01 : Proceedings of the 21st international conference on distributed computing systems, p 635, Washington, DC, USA, 2001, IEEE Computer society.

Cheung-Foo-Wo.D, Blay-Fornarino.M, Tigli.J.Y, Lavirotte.S, Riveill.M: «Adaptation dynamique d'assemblage de dispositifs par des modèles».2ème journée sur l'ingénierie dirigée par les modèles, 2006 a.

Cheverst.K, Mitchell.K, Davies.N: «The role of adaptive hypermedia in a contextaware tourist guide». Communications of the ACM, Vol 45, N°5, pp 47-51,2002.

Colombo.M, Di Nitto.E, Mauri.M: «Scene: A service composition execution environment supporting dynamic changes disciplined through rules». In A.Dan, W.Lamersdof (Editions), Service-oriented computing, ICSOC'06, Vol 4294 of the lecture Notes in computer science. Springer, Berlin, Heidelberg, pp 191-202, 2006.

Coppolino.L, Romano.L, Mazzocca.N, Salvi.S: «Web services workflow reliability estimation through reliability patterns». Security and privacy in communication networks and workshops, 2007.

CORBA Component model specification OMG Available specification Version 4.0 forma 1.

Cremene.M, Riveill.M, Loghin.C, Miron.C : « Adaptation dynamique de services ». In proceedings of the 1st conférence francophone sur le déploiement et la reconfiguration de logiciels, Grenoble, France, 2004.

Cremene.M, Riveill.M, Martel.C : « Towards unanticipated dynamic service adaptation ». In proceedings of the 3rd international workshop on coordination and adaptation techniques for software entities, pp 25-34, Nantes, France, 2006.

Crnkovic.I,Hnich.B, Jonsson.T, and Kiziltan.Z : « Spécification, implementation, and deployment of components ». Communications of the ACM. October 2002, Vol.45, N°10.

David.P.C, Ledoux.T: «A generic framework for context-aware applications». In S.Terzis and D.Donsez, Editors, MPAC, Vol 115 of ACM international conference proceedings series, pp 1-7, 2005.

Demeure.I, Duda.A: «Systèmes répartis et réseaux adaptifs au contexte (contextaware». 2003.

Dey.A.K: «Understanding and using context». Personal and ubiquitous computing, Vol 5, N°1, pp4-7, 2001.

Dockhm Costa.P : « Architectural support for context-aware applications : from context models to services platforms ». PhD thesis, center for telematics and information, University of Twente, The Netherlands, 2007.

Dodani.M.H : « From objects to services : a journey in search of component reuse nirvana ». Journal of object technology, Vol 3, N°3, pp 49-54, 2004.

Donsez.D, Gibello.P.Y : «Advanced transaction models for EJB and web services ». 2001.

Dowling.J, Cahill.V : « Self-managrd decentralised systems using K-components and collaborative re-inforcement learning ». In WOSS'04 : Proceedings of the 1st ACM Sigsoft workshop on self-managed systems, pp 39-43, New York, NY , USA, 2004.

Dieng.A.I, 2010 : Selecta : « Une approche de construction d'applications par composition de services », Thèse de doctorat de l'Université de Grenoble, Septembre 2005.

Dietze.S, Gugliotta.A, Domingue.J : »Bridgine the gap between mobile application contexts and semantic web resources ». In book of context-aware mobile and ubiquitous computing for enhanced usability : Adaptive technologies and applications, pp 217-234, Hershey-New York, 2009. Information science reference.

Dustdar.S, Shreiner.W : « A survey on web services composition ». Inj.web and grid services, Vol 1.1, N°1, 2005.

Efstratiou.C, Friday.A, Davies.N, Cheverst.K : « Utilizing the event calculus for policy driven adaptation on mobile system ». In Policy, pp 13-24, 2002, IEEE Computer society.

Eheierman.O, Cook.J.D : « Improving home automation by discovering regularly occurring devices usage patterns». In proceedings of the 3rd IEEE international conference on data mining, p 537, Washington, DC, USA, 2003. IEEE Computer society.

El Boussaidi.G, Mili.H : « Les langages de description d'architectures ». Latece technical report, October 2006.

Erradi.A, Maheshwari.P: «WSBUS: QoS-aware middleware for reliable web services interactions». In proceedings of the 2005 IEEE international conference on etechnologie, e-commerce and e-service (EEE'05). IEEE computer society, Washington, DC, USA pp 634-639, 2005.

Fiadeiro.J.L, Lopez.A, Bocchi.L : «A formal approach to service component architecture». In Proceedings of the 3er international workshop web services and formal methods, LNCS, Vol 4184, Springer-verlag, pp 193-213, 2006.

Fleurey.F, Solberg.A: «a domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems». In proceedings of the 12th international conference on model driven engineering languages and systems. MODELS'09. Springer-verlag, Berlin, Heidelberg, pp 606-621, 2009.

Gadellini.V, Casalicchio.E, Grassi.V, Lo Presti.F: «Adaptive management of composite services under percentile-based service level agreements». In P.Maglio, M.Weske, J.Yang. M.fantinato (Editions), service-oriented computing, Vol 6470 of Lecture Notes in computer science. Springer, Berlin, Heidelberg, pp 381-395, 2010.

Garlan.D : « Software architecture : a road map ». In the future of software engineering, A.Finkekstein (ed), ACM press, 2000.

Garlan.D, Monroe.R.T, Wile.D : « Acme : Architectural description of component based-systems ». Foundations of component-based systems, G.T Leavens and M.Sitaraman (eds), Cambridge University Press, 2000.

Garlan.D, Cheng.S.W, Huang.A.C, Shmerl.B, Steekiste.P : « Raibow : Architecturebased self-adaptation with reusable infrastructure ». Computer, Vol 37, N°10, pp 46-54, 2004.

Giovanni.A, Vinvenzo.L, Michele.N, Stefano.R : « Ambient intelligence Framework for context-aware adaptive applications ». In Proceedings of the 7th international workshop on computer architecture for machine perception, pp 327-332, Washington, DC, USA, 2005. IEEE Computer society.

Grodin.G : « Un modèle d'agents auto-adaptables à base de composants » PhD Thesis, Ecole nationale supérieure des mines de Saint-Etienne, France, 2009.

Hachette. Le dictionnaire universel francophone en ligne. http://francophonie.hachette-livre.fr.

Hadjab.S. Assemblage de composants logiciels : « Les composants-composites », DEA réseaux avancés de connaissances et organisations, Université de technologie de Troyes, Juillet 2003.

Haifeng.Y, Vahdat.A : « Design and evaluation of a conit-based continuous consistency model for replicated services. ACM Trans.Comput.Sys. Vol 20, pp 239-282, August 2002.

Heineman.G.T : « A model for designing adaptable software components ». Sigsoft softw.Eng.Notes. Vol 25, N°1, pp 55-56, 2000.

Heineman.G.T, Councill.W.T,Component-based software engineering : Putting the pieces together (ACM Press). Addison-Wesley Professional, June 2001.

Huhns.M.N ,Singh.M.P : « Service-oriented computing : key concepts and principles ». IEEE Internet computing, Vol 9, N°1, pp 75-81, 2005.

Jaeger.C.M: «Optimising quality of service for the composition of electronic services». PhD thesis, Berlin University, Germany, 2007.

Jaga.I, Seng Wai.L, Rakotonirainy.A, Varuni.W, Zaslavsky.A.B: «An open architecture for pervasive systems». In proceedings of the 3rd international working conference on new developments in distributed applications and interoperable systems. Pp175-188,Deventer, The Netherlands, 2001.

Kang.Z, Wang.H, Hung.P.C.K : « WS-CDL+ : An extented WS-CDL execution engine for web service collaboration ». In proceedings of the IEEE international conference on web services (ICWS'07), July 2007, Salt Lake city, Utah, USA, pp 928-935.

Kang.Z, Wang.H, Hung.P.C.K : « WS-CDL+ for web service collaboration ». Information systems frontiers, 2007, Vol 9, N°4, pp 375-389.

Kavantzas.N, Burdett.D, Ritzinger.G, Fletcher.T, Lafon.Y, Barreto.C : « Web service choreography description language ». Version 1.0 W3C candidate recommendation, 2005.

Kephart.J.O, Chess.D.M : « The vision of autonomic computing ». Computer, Vol 36, N°1, pp 41-50, 2003.

Ketfi.A. « Une approche générique pour la reconfiguration dynamique des applications à base de composants logiciels ». Thèse de doctorat de l'université Joseph Fourier de Grenoble, Décembre 2004.

Ketfi.A, Belkhatir.N, Cunin.P : « Adaptation dynamique concepts et expérimentation ». ICSSEA'02.

Koning.M, Sun.C-A, Sinnema.M, Avgerou.P: «VxBPEL: Supporting variability for web services in BPEL». Information and software technology, Vol 51, pp 258-269, February 2009.

Kouici.N, « Gestion des déconnexions pour applications réparties à base de composants en environnements mobiles ». Thèse de doctorat en informatique, institut national des télécommunications (INT), Université d'Evry Val d'Essonne, France, 16 Novembre 2002.

Kung-Kiu.L : « Software component models ». ICSE'06, May 20-28, Shanghai, China, 2006.

Layaida.O, Hagimont.D : « Designing self-adaptative multimedia applications through hierarchical reconfiguration ». In Lea Kutvonen and Nancy Alonistioti, editors, DAIS, Vol 3543 of lecture Notes in computer science, pp 95-107, Springer, 2005.

Lemlouma.T, Layada.N : « Content adaptation and generation principles for heterogeneous clients ». In W3C Workshop on device independent authoring techniques, September 2002, St Leon-Rot, Germany.

Leymann.F : « Web service flow language ». Version 1.0 IBM report, 2001.

Louis.S.J, Shankar.A : « A context learning can improve user interaction ». In proceedings of the IEEE international conference on information reuse and integration, pp 115-120, 2004.

Luckham.D.C, Kenney.J.J, Augustin.L.M, Vera.J, Bryan.D, Mann.W : « Specification and analysis of system architecture using Rapide ». IEEE Transactions on software engineering. Vol 21, N°4, April 1995.

Macedo De Amorim.K, « Modélisation d'aspects qualité de service en UML : Application au composants logiciels ». Thèse de doctorat de l'Université de Renne 1, Mai 2004.

Magee.J, Dulay.N, Eisenbach.S, Kramer.J : «Specifying distributed software architectures ».The 5th European software engineering conference ESEC'95, Spain, 26 September 1995.

Malloy.B.A, Kraft.N.A, Hallstrom.O and Voas.J.M : « Improving the predictable assembly of service-oriented architectures ». 2006, IEEE software, Vol 23, IEEE Computer society press.

McBurney.S, Howard . W, Taylor.N ,Papadopoulou.E : « Managing user preferences for personalization in a pervasive service environment ». Advanced international conference on telecommunications, p: 32, 2007.

McBurney.S.M, Papadopoulou.E, Taylor.N, Howard.W : « Adapting pervasive environments through machine learning and dynamic personalization ». In Proceedings of the 28th IEEE international symposium on parallel and distributed processing with applications, pp 395-402, Washington, DC, USA, 2008.IEEE Computer society.

McKenzie.M, Laskey.K, McCabe.F, Brown.P, and Metz.R : « Reference model for service-oriented architecture ».1.00Technical report wd-soa-rm-cd1, Oasis, February 2006.

McKinley. P.K, Sadjadi. S.M, Kasten. E.P, Cheng. B.H.C : « Composing adaptive software ». computer Vol 37, pp 56-64, July 2004.

Mehta.N, Medvidovic.N, Phadke.S : « Towards a taxonomy of software connectors ». ICSE 2000, Limerick-Ireland.

Melliti.T : « Interopérabilité des services web complexes, application aux systèmes multi-agents ». Thèse de doctorat de l'Université de Paris Dauphine, France, Décembre 2004.

Moser.O, Rosemberg.F, Dustdar.S: « Non-intrusive monitoring and service adaptation for WS-BPEL». In Proceedings of the 17th international conference on WWW'08. ACM, new York, NY, USA, pp 815-824, 2008.

Mosincat.A, Binder.W: «Transparent runtime adaptability for BPEL processes». In Proceedings of the 6th international conference on service-oriented computing. ICSOC'08. Springer-verlag, Berlin, Heidelberg, pp 241-255, 2008.

Motahari Nezhad.H.R, Benatallah.B, Casati.F, Toumani.F : « Web services interoperability specifications ». 2006, IEEE computer, Vol 39.

Nano.O : « Un modèle de réécriture pour l'intégration des services ». Thèse de doctorat de l'université de Nice, Sophia Antipolis, Décembre 2004.

Narenda.N.C, Ponnalagu.K, Krishnamurthy.J, Ramkumar.R: «Runtime adaptation of non-functional properties of composite web services using aspect-oriented computing». ICSOC'07. Springer-verlag, Berlin, Heidelberg, pp 546-557, 2007.

Nayak.R, Tong.C : « Applications of data mining in web services ». In web information systems, WISE, Lecture notes in computer science, pp 199-205, Springer, 2004.

NewMarch.J, A Programmer's Guide to Jini Technology. A Press, 2001.

Oasis : « Reference model for service oriented architecture ». October 2006.

Oliveira.N, Barbosa.L.S: « A self-adaptation strategy for service-based architectures». In the 8th Brazilian symposium on software components, architectures and reuse. SBCARS'14, pp 44-53. IEEE 2014.

OSGI alliance-The dynamic module system for Java, 2010. Release 4.

Osman.T, Thakker.D, Al-Dabass.D : « Bridging the gap between workflow and semantic-based web services composition ». In Proceedings of the web service composition workshop WSCOMPS'05.

O'Sullivan.J, Edmond.D, and Ter Hofstede.A : « What's in a service ? ». Distrib.Parallel databases,12(2-3),pp117-133, 2002.

Papazoglou.M.P : « Service-oriented computing : concepts, characteristics and directions ». Web information, 2003. WISE 2003. Proceedings of the fourth international conference on, pp 3-12, December 2003.

Papadopoulou.E, McBurney.S, Taylor.N, Howard.W : « A dynamic approach to dealing with user preferences in a pervasive system ». In Proceedings of the 2008 IEEE International symposium on parallel and distributed processing with application, pp 409-416, Washington, DC, USA, 200. IEEE Computer society.

Papazoglou.M.P ,Van Del Heuvel.W : « Service-oriented architectures : approaches, technologies and research issues ». The VLDB Journal, Vol 16, N°3, pp389-415, July 2007.

Pascoe.J, Ryan.N, Morse.D.R: «Using while moving: Hci issues in-eldwork environments». ACM Trans. Comput.-Hum.Interact., Vol 7, N°3, pp 417-437, 2000.

Peltz.C : « Web services orchestration and choreography ». IEEE computer, 2003, Vol 36, N°10, pp 46-52.

Pignotti.E, Edwards.P, Grimnes.G.A : « Context-aware personalized service delivery ». In proceedings of the European conference on artificial intelligence, pp 1077-1078, 2004.

Preuveneers.D, Victor.K, Vanrompay.Y, Rigole.P, Kirsch.M, Berbers.Y : « Context-aware adaptation in an ecology of application ». In book of context-aware mobile and ubiquitous computing for enhanced usability : adaptive technologies and applications, pp 1-25, Hershey, New York, 2009. Information science reference.

Projet Accord. Assemblage de composants par contrat en environnement ouvert et réparti. Livrable1.1-1 Juin 2002.

Projet Accord : « Assemblage de composants par contrat en environnement ouvert et réparti : le modèle de composants Corba », livrable 1.1-4, Mai 2002.

Projet Rntl Faros : « Etat de l'art sur la contractualisation et la composition ». Livrable F-1-1, p10, Août 2006.

Qianhui.L, Chung.J.Y : « Analyzing service usage patterns : methodology and simulation ». In Proceedings of the IEEE international conference on e-business engineering. pp 359-362, Washington, DC, Usa, 2007, IEEE computer society.

Quan.W, Jianmin.H : « Personalized recommendation services based on serviceoriented architecture ». In proceedings of the 2006 IEEE Asia-Pacific conference on services computing. Pp 356-361, Washington, DC, USA, 2006. IEEE computer society.

Rakic.M, Medvidovic.N : « Increasing the confidence in off-the-self components : A software connector-based approach ». SSR'01, May 18-20, 2001, Toronto, Ontario, Canada.

Ramaparany.F, Benazzouz.Y, Bel Martin.M : « Agenda driven home automationtowards high level context aware systems ». Symposium and workshops on ubiquitous, autonomic and trusted computing, pp: 125-130, 2009.

Raibulet.C: «adaptive resource and management in a mobile enabled environment». In book of context-aware mobile and ubiquitous computing for enhanced usability: Adaptive technologies and applications, pp 235-257, Hershey-New York, 2009. Information science reference.

Reinhard.O, editor : « Adaptive user support : ergonomic design of manually and automatically adaptable software ». L.Erlbaum Associates Inc, Hillsdale, NJ, USA, 1994.

Rosenberg.F: «QoS-aware composition of adaptive service-oriented systems». PhD thesis, Technical University Vienna, Austria, 2009.

Saowanee.S : « Context-based service adaptation platform improving the user experience towards mobile location services ». In proceedings of the international conference on information networking. Vol 23, 25, pp 1-5, 2008.

Schilit.B.N, Theimer.M.M: «Disseminating active map information to mobile hosts». In IEEE network, special issue on nomadic computing, Vol 8, N°5, pp 22-32, 1994.

Schilit.B.N, Adams.N, Want.R: «Context-aware computing applications». In Proceedings of the IEEE workshop on mobile computing systems and application, pp 85-90, Santa Cruz, USA, 1994.

Senart.A : « Canevas logiciel pour la construction d'infrastructures logicielles dynamiquement adaptable ». Thèse de doctorat de l'institut national polytechnique de Grenoble, France, Novembre 2003.

Serena.F, Senart.A, Siobhan.C: «Addressing dynamic contextual adaptation with a domain-specific language». In Proceedings of the 1st international workshop on software engineering for pervasive computing applications, systems, and environments. p2, Washington, Dc, USA, 2007. IEEE Computer society.

Shaw.M, De Line.R, Klein.D.V, Ross.T.L, Young.D.M, and Zelzsnik.G : « Abstractions for software architecture and tools to support them ». IEEE Transactions on software engineering. Vol 21, N°4, April 1995.

Shin.M, Cooke.D : « Connector-based self-healing mechanism for components of a reliable system ». DEAS 2005, May 21, 2005, St Louis, Missouri, USA.

Shulte.W.R ,Natis.Y.V : « Service oriented architectures ».Part1&2, April 1996, Gartner Group.

Smiee.A.J : »Customer adaptive communications services ». In proceedings of the 10th IEEE region conference TENCOM, pp 886-890, Melboune, Australia, 1992.

Sonntag.M, Karasoyanova.D: « Compensation of adapted service orchestration logic in BPEL'n'aspects». In Proceedings of the 9th internatuinal conference on business process management (BPM'11). Springer-verlag, Clermont-Ferrand, France, pp 1-6, 2011.

Sun Microsystems. "Enterprise JavaBeans Technology".

Sun.H, Wang.X, Zhou.B, Zou.P : « Research and implementation of dynamic web services composition ». APPT 2003, LNCS 2834, Springer-Verlag, Berlin Heidelberg, pp 457-466.

Szyperski.C, Gruntz.D, and Murer.S, Component software : Beyond object-oriented programming. Addison-Wesley Professional (2nd edition), England, 2002.

Taylor.D.A, Object technology (2nd edition) : a manager's guide. Addison-Wesley Longman Publishing Co., Boston, MA, USA, 1998.

Thatte.S : « Xlang : Web service for business process design ». Microsoft Specification, 2001.

Tigli.J.Y, Cheung-Foo-Wo.D, Lavirotte.S, Riveill.M: «Adaptation au contexte par tissage d'aspects d'assemblage de composants déclenchés par des conditions

contextuelles». RSTI serie ISI, Adaptation et gestion de contexte, Vol 11, N°5, pp 89-114, 2006.

Tilevich.E, Smaragdakis.Y : « J-orchestra : Enhancing java programs with distribution capabilities ». ACM Trans.Softw.Eng.Methodol, Vol 19, N°1, pp 1-40, 2009.

Tournier. J. Q: « Une architecture à base de composants pour la gestion de la qualité de services dans les systèmes embarqués mobiles ». Thèse de doctorat de l'institut national des sciences appliquées de Lyon, Juillet 2005.

Vallée.M, Ramparany.F, Vercouter.L : « Using device services and flexible composition in ambient communication environments ». In proceedings of the 1st international workshop on requirements and solutions for pervasive software infrastructure, Dublin, Ireland, 2006.

Wohed.P, Van Der aalst.W, dumas.M, Ter Hofstede.A : « Analysis of web services composition languages : the case of BPEL4WS ». In Proceedings of the 22nd international conference on conceptual modeling. October 2003, pp 200-215, Chicago, IL, USA. LNCS Springer 2003.

Wynen.F : »Conception et développement d'une plate-forme de coopération interorganisationnelle : le point de synchronisation ». Mémoire d'ingénieur. CNAM, Nancy, 2003.

Yahiaoui.N, Traverson.B, levy.N : « Classification and comparison of adaptable platforms ».2004.

Zaidenberg.S : »Apprentissage par renforcement de modèles de contexte pour l'informatique ambiante ». PhD thesis, Grenoble INP, France, 2009.

Zhang.C, Zhang.Z : « Trading replication consistency for performance and availability : an adaptive approach ». In 23th international conference on distributed computing systems (23th ICDCS'03), pp 687-695, providence, May 2003. IEEE computer society.

Résumé

Depuis plusieurs années les systèmes informatiques ont émergé, ils occupent désormais une place dans notre vie quotidienne. Les besoins grandissants et sans cesse croissants des utilisateurs des systèmes informatiques, ont engendré le développement d'applications de plus en plus grandes et complexes. La complexité de ces applications pose des problèmes tels que la réutilisation, l'installation, l'administration, l'évolution et l'adaptation dynamique.

Notre travail propose une architecture une architecture logicielle qui rend possible l'adaptation dynamique de services construits par assemblage de composants, en fonction de contextes d'utilisation variés.

Mots clés : Service, adaptation dynamique, composant.

Abstract

For several years computer systems have emerged, they now have a place in our daily lives. The growing needs and constantly growing users of computer systems have led the development of applications increasingly large and complex. The complexity of these applications poses problems such as reuse, installation, administration, development and dynamic adaptation.

Our work proposes an architecture software architecture that enables the dynamic adaptation of services built by assembling components, depending on a variety of use environments.

Keywords: Service, dynamic adaptation component.

الملخص:

منذ عدة سنوات ظهرت أنظمة الكمبيوتر، تحتل الآن مكانا هاما في حياتنا اليومية. وقد أدت الاحتياجات المزدادة و المتزايدة لمستخدمي نظام الحاسوب إلى تطوير تطبيقات كبيرة و معقدة على نحو متزايد.

تعقيد هذه التطبيقات يطرح مشاكل مثل إعادة الاستخدام، التركيب، الإدارة، التطور و التكيف الديناميكي.

يقترح عملنا هذا، بنية البرمجيات التي تجعل من الممكن التكيف الديناميكي لخدمات مبنية على أساس تجميع مكونات برمجية، وفقا لسياقات مختلفة من الاستخدام.

الكلمات الرئيسية: الخدمة، التكيف الديناميكي، المكون.