Context-Aware Service Adaptation: An Approach Based on Fuzzy Sets and Service Composition

MOHCINE MADKOUR, DRISS EL GHANAMI, ABDELILAH MAACH AND ABDERRAHIM HASBI

Computer Science Department
Mohamadia School of Engineers
Mohamed 5 University
Ibn Sina, Rabat, 765 Morocco

With the pervasive use of mobile devices and the need for ubiquitous computing, the issue of “context” now becomes a hot topic in human computer interaction research and development. Further, the interface of the interaction beyond the desktop is moving from humans vs. computers to humans vs. context-aware environments. This leads the context to constitute an underlying part of service behavior, especially when interaction with end-users is involved, and consequently this demands the context-aware computing systems to be designed to automatically adapt its behavior to changing environment. Thus, pervasive computing applications need to be more autonomous and sensitive to context.

However, in real-life systems context information is naturally dynamic, vague and complex, which lead to an inexact match between provided and required service capabilities. In this vision, we propose in this paper a three-phases adaptation approach: firstly we select the suitable services to the current context and we recommend them to the adaptation process, in the service adaptation phase we perform adaptation by using fuzzy sets represented with linguistic variables and membership degrees to define the user’s context and the rules for adopting the policies of implementing a service. Finally we deal with the complex requirements of the user by the composition of the obtained adaptable atomics services.

Keywords: context, context-awareness, service selection, service adaptation, service composition, fuzzy sets, AI planning

1. INTRODUCTION

The term Ubiquitous Computing, introduced by Weiser [1], refers to the seamless integration of devices into users’ everyday life. This term represents an emerging trend towards environments composed of numerous computing devices that are typically mobile or embedded and that are connected to a network infrastructure composed of a wired core and wireless edges. The pervasive computing, foreseen by Ubiquitous Computing, is a promising computing paradigm by which users can access resources they need anywhere at anytime. Pervasive computing also emphasizes on the technique invisible, i.e., users submit their requirements and get the results through access points, without knowing how to achieve those requirements.

Most existing approaches to context-aware systems propose either a user-centric or a service-centric view of context. User-centric approaches promote applications that move with the users and follow their preferences. Service-centric approaches promote
service adaptability to the context changes. We propose a solution that combines both views. Indeed, both users-related and services-related contextual requirements are taken into account in our approach to service composition.

More specifically, in ubiquitous environments, context information is naturally uncertain and fuzzy and hence an important issue to address in designing a context-aware computing is how to effectively select services for adaptation according to the user’s current context. So, how service selection can proceed to choose a corresponding services that match best the current context situation taking into account its uncertainty, such as recommender systems. Some elements of response can be found in the representation of the context model that may be designed in a way that allows the representation of uncertainty, i.e., the context model may be extended by elements from fuzzy theory.

Once the services are selected, for ensuring an acceptable QoS and allowing for adaptation to changes in the operating environment, the next phase consists of choosing, among recommended services, the most adapted to the current context, we define the policy-based adaptation of context-aware service as the automatic selection of the best policy for delivering the service.

On the other hand, generally on current service-oriented applications, individual context-aware service usually cannot meet the requirements arising from real world environment and applications; a single service is based on one single resource which is relatively simple, while the complicated requests are usually complex and can be achieved by composing the single services. Service composition is a good method to satisfy dynamic and complicated requests, but, there are still two problems that need to be resolved, (i) how to synthesize the composite services according to the requests, and (ii) how the composite service should be run in realtime and recursively.

In our opinion, to establish a generic context-aware adaptation approach, we need:

- Models and tools to describe the adaptation source (i.e. context),
- Models and tools to effectively select services according to the user’s context,
- Adaptation policies putting all the previous models and tools together,
- Models and tools to achieve service composition.

In this paper, we perform the adaptation process by using fuzzy sets to define user’ context and rules for adopting the policies of implementing a service, thereafter we deal with complex user’ requirements by composing adaptable atomics services.

In the remainder of this paper, section 2 discusses some related works. Section 3 presents an example scenario; section 4 describes our selection method of context-aware services. Section 5 addresses policy-based context-aware service adaptation. Section 6 presents our AI planning service composition and finally section 7 concludes the paper.

2. RELATED WORK

The related work of our work touches the related works of three researches contexts: service selection/recommendation, service adaptation and service composition.

Most research tracks on context-aware services focus on how to describe and discover these services. Therefore, applying context-aware techniques to retrieve services
has gained lots of attentions: Yang et al. [2] design an event-driven rule based system to recommend services according to people’s context changes. They provide an ontology-based context model to represent context and utilize the context to assist service discovery. Ben Mokhatar et al. [3] propose the use of ontologies in Semantic Web Ontology Language (OWL-S) for the semantic description of functional and non-functional features of services in order to automatically and unambiguously discover such services.

However, researches cited above do not consider inexact retrieval caused by fuzzy or uncertain context information. Context information is naturally vague and uncertain. Uncertainty in context information is traditionally handled by appropriate models, such as proposed by Chalmers et al. [4], who represent context values by intervals or sets of symbolic values. Other approaches such as [5] deal with uncertainty using fuzzy logic. However, they do not concentrate on inexact service discovery. Service selection algorithms ought to consider uncertainty represented in their models. We argue that fuzziness on context information must be considered when selecting services.

In literature, many platforms were proposed to facilitate adapting context-aware services. Among these platforms, we can cite: The Chisel system [6] introduced a dynamic services adaptation framework which decomposes the particular aspects of a service object into multiple possible behaviors. Whenever the context information changes, the service object will be adapted to use the different behaviors according to the adaptation policy. Policy-driven Mobile Agents [7] and Case-Based Situation Assessment [8] are also proposed for adaptation in context-aware system.

In the other hand, composing services together is the new challenge awaiting the Service Oriented Architecture (SOA) middleware meeting the pervasive environments [9]. Indeed, the variety of service providers in a pervasive environment, and the heterogeneity of the services they provide require applications and users of these kinds of environments to develop models, techniques and algorithms in order to compose services and execute them. The service composition needs to follow some requirements in order to resolve the challenges brought by pervasively.

Several surveys dealt with service composition. Many of them classified the middleware under exclusive criteria such as manual versus automated, static versus dynamic, and so on. Others [10] classified the service composition middleware under different domains such as artificial intelligence, formal methods, and so on. But none of these surveys proposed a generic reference model to describe the service composition middleware in pervasive environments. Existing composition architectures in the literature predominantly accommodate components described in XML-based languages. SAHARA [11] and Anamika [12] are examples of distributed composition architectures; all others follow a centralized approach. Early works, such as eFlow [13] employs graph structures to model composite services, while the more recent work uses DAML-S [14] structures to store composite service templates.

Although a rich landscape in adaptation related researches, a complete and generic context-aware adaptation approach is still missing. The existing solutions are generally proposed to incrementally create adaptive resources (service-centric) or adapting software components to meet user context (user-centric). They are not suitable for both user and service centric adaptation; in our approach, adapting is selecting the best policy for service implementing and recursively creating new services (composites services) to new utilization contexts.
3. EXAMPLE SCENARIO

To illustrate how context-aware services can be built using the proposed adaptation method, this section introduces the following scenario.

Use Case 1: A user, called Mohcine, subscribes to a mobile network provider, which hosts an infotainment portal. This portal offers users a broad array of resources and services, such as point of interest information, on-line shopping, and search engines. It is a single starting point for retrieving information from multiple, diverse sources. In this first case, Mohcine is using his SmartPhone while walking around Market Square in his city. Mohcine has a subscription to an infotainment portal, available from his local mobile provider. This provides Mohcine access to a restaurant recommendation service, for example, to make lunch plans with his college friends. The portal uses adaptation framework to personalize the service to requester’ context, then for non-atomic services the portal uses the composition framework, which assembles a composite service to deal with Mohcine’s request. The resulting service, tailored to help Mohcine locate a Lebanese restaurant, is composed from atomic services, such as RestaurantFinder, a city-based restaurant directory, and DirectionsFinder, the navigation service.

Use Case 2: Later in the day, Mohcine lands in the suburb. He wishes to pick up his friend Reda in the airport. In his car Mohcine is offered the usage of the Navigation Service already preconfigured with data received from Mohcine Calendar Service that contains an entry with Reda arrival date. As Mohcine is driving, he would prefer the airport directions to be routed to his in-vehicle information system (IVIS), so he uses the Navigation Service that interacts with a Traffic Information Service in order to calculate the fastest route to the airport. When he approaches the airport zone, his IVIS discovers a new service, the Airport Information Service. This service provides, among others, the information about arrival time and gate number of Reda’s flight. The Navigation Service automatically uses the recently discovered service and calculates a new route to the given gate while Traffic Information Service provides the information about available parking places near the gate. Mohcine parks the car and arrives at the gate just in time to pick up his friend. A while when Mohcine wants to consult his mail, he discover that he has only the mail header and a sign indicating that the receiving process is not finished and will continue when a higher network bandwidth and other resource are available.

In the first case, Mohcine has the goal to get direction to a chosen place, and his context is enriched by the user' subscription to the needed service, his request result in the composite service being constructed from two atomic services tailored to the user’ context. Despite In the second case, Mohcine does not know about the types of services that may have the services providers work as a service recommender and predict the service based on the situation context; Due to changing service availability, when Mohcine reaches the airport zone the existing composition is altered and another service, the Airport Information Services, is added. In the last, the email service has a possibility to make a suitable choice among five policies: headMail, fullMail, EncryptedMail, BigMail and EncryptedBigMail based on the assessment of the current contextual situation.
4. CONTEXT-AWARE SERVICE SELECTION

In recommendation systems, meeting user requirements involves a thorough understanding of their interests expressed explicitly through search engine queries or implicitly through browsing behavior and search context. We consider that the user is not always able to describe the suited services due to the dynamic change of his context. Nevertheless, context information is naturally uncertain and incomplete due to the uncertainty of measuring or to the fuzziness in elucidation. It is the goal of fuzzy theory to formalize the approximate reasoning that is used by humans in everyday life. It follows the human functioning of representing values as terms instead of numeric values. In this section, we describe our method to model the user's context to meet individual user needs.

Based on the basic concepts such as User, Calendar, Device, Time and Place, we have developed an ontology-based context model. The ontology has been developed by using the Web Ontology Language, a W3C standard well-supported in semantic engines.

In order to handle fuzzy information in OWL model ontology, we established a representation pattern. The pattern is applicable to properties that are interrelated to the same base variable and to the same pair of concepts. For instance, let us consider the base variable distance, and the concepts User and Place, we can establish properties like User is-close-to a Place or User is-far-from a Place. The presence of each property depends on the membership of the distance value to a prefixed interval. For example, considering the first interval as LowDistance = 0-10 meters, it can be said that User is-close-to depends on LowDistance, more formally is-close-to|LowDistance. Fig. 1 (a) shows an abstract representation of this mechanism for a series of n properties and related n intervals. Here, concepts have been enclosed in oval shapes, whereas properties are represented by arrows. In order to capture vagueness in this representation, we propose the extension shown in Fig. 1 (b). Here, an OWL group of properties is transformed into a concept, which includes a specification of the degree for each property. In other words, we assert that there is a property with a certain degree. Each degree is the membership level of the base variable to a specific fuzzy set.

It is worth noting that this scheme can be used also in case of a property related to a single concept. In such case, the concept property corresponds to the concept itself. In
Fig. 2, the user context ontology is presented. This ontology is made of 10 general concepts and 25 properties, together with 5 concepts and 14 properties for the fuzzy representation. In particular, general concepts such as Time and Place are inherited from publicly available ontologies according to the best practices of reusing domain ontologies. In Fig. 2, such external ontologies are enclosed in dashed rectangular shapes. Concepts are connected by properties, represented with directed black edges in the figure. Edges with white arrowhead show classical inheritance (i.e., an is-a relation).

The model comprises a set of rules to infer the current situations on the basis of the situation ontology. Rules are expressed in the semantic web Rule Language SWRL, an emerging standard that extends OWL with additional rule-based knowledge representation. In terms of expressiveness, this reasoning standard corresponds to description logics, a particular decidable fragment of first order logic, and is named OWL DL. Fig. 3 shows an example of rule in human readable syntax (a), commonly used in the literature, and in natural language (b). We point out that there are two types of antecedent conditions, i.e., crisp (binary) and fuzzy, represented in Fig. 3 (b) in bold and italic bold, respectively. The conditions is a participant and has type are derived from the user’s calendar, and are inherently crisp, whereas the other conditions can be assessed only with vagueness. This also implies that also the conclusion inferred from the rule is characterized by vagueness. This vagueness can be represented directly in SWRL, which implements some mechanisms to express truth degrees and related membership functions.

IF user1 IS A PARTICIPANT to the scheduled event AND user1 IS moving AND user1Time IS BEFORE the scheduled event start-time AND event HAS TYPE business THEN user1 IS IN A SITUATION OF pre-meeting-on-movement

Fig. 3. A rule example.
Once some situations have been inferred, with a certainty degree, a task ontology allows connecting a situation to specific tasks, and then specific tasks to specific service to be recommended. Furthermore, such resources are tailored by proper contextual information, selected according to the identified user task. In Fig. 3 (c) the discovery service ontology is represented.

5. OUR ADAPTATION MODEL

In this section, we detail our functional adaptation approach based on policy of delivering the service. We detail each step in the remaining parts of this section.

5.1 The Fuzzy-based Functional Adaptation

The proposed Fuzzy-based Functional Adaptation (FFA) is similar to the traditional fuzzy control process (Fig. 4) in that it is in the following three steps:

Step 1: Fuzzifier. Each context information is represented by a linguistic variable, which may be associated with several linguistic values. Each linguistic value is represented by a predefined application-related membership function (e.g., $\mu_{\text{Network_{maxRateHigh}(x)}}$, $\mu_{\text{Network_{maxRateLow}(x)}}$, where $x$ is the crisp value of the corresponding linguistic variable). By using the membership functions we translate the input context crisp value into a set of pairs, each consisting of a linguistic value and a membership degree for that value. The fuzzified context information is then combined into a fuzzified context situation (see definition later).

Step 2: Calculation of fitness degree for each policy. In context-aware computing, a service can be delivered using several policies and each policy is associated with a particular context situation. We assume that a service can be delivered using only one policy at any time. During the inference process, each policy will be assigned a fitness degree indicating to what degree the policy is suitable for being used under the current context situation. The fitness degree is assigned by using a fitness function, which calculates the fuzzy distance between the policy’s most suitable context situation and the current context situation. The fitness degree will decrease as the fuzzy distance increases. The most suitable context situations and some additional intervention rules (e.g. application profile, user preference and system feedback) form the rule base of our fuzzy-based process.

Step 3: For a requested service, the policy with the largest fitness degree will be selected as the best policy for delivering the service under the current context situation.
5.1 Formulized Solution

To introduce the proposed formulized solution, we first stretch the definition of the concepts and terminologies used in [17] to our Fuzzy-based Functional Adaptation Process FFAP, and then design fitness functions base on the concepts.

**Definition 1 (Service):** A service represents a function that is provided by the middleware and invoked by a mobile application. Let \( S = \{S_1, S_2, S_3, \ldots, S_q\} \) be the set of services provided by the middleware, where \( S(i) \leq q \) represents the \( i \)th service, \( q \) is the number of services. We use \( S_{\text{need}} \) to denote the set of services requested by the mobile application.

**Definition 2 (Policy):** A policy represents a method used to deliver a service with a certain resource requirement and quality-of-service condition. Let \( P = \{p_1, p_2, p_3, \ldots, p_m\}, i \in [1, q] \) be a set of policies which can be adopted for delivering the \( i \)th service \( S_i \) (\( \forall S_i \in S \)) where, \( p_i(j) \leq m \) represents the \( j \)th policy of \( S_i \), \( m \) is the number of all policies for \( S_i \).

**Definition 3 (Context):** Let \( C = \{C_1, C_2, C_3, \ldots, C_n\} \) be a set of context which are monitored by the middleware, where \( C(i) \leq n \) represents the \( i \)th context information, \( n \) is the number of all monitored context.

**Definition 4 (Context Situation):** A context situation is a combination of context information. Let \( LV = \{Lv_1, Lv_2, Lv_3, \ldots, Lv_k\} \) be a set of linguistic values. The context situation at time \( t \) is denoted by \( SI(t) \) and represented by a set of 3-element tuples:

\[
SI(t) = \{(c_a, lv_b, \mu_{c_a,lv_b}, \text{value}_of(c_a, t))|c_a \in C, a \in [1, n], lv_b \in LV, b \in [1, k]\}
\] (1)

where, \( c_a, 1 \leq a \leq n \) is certain context information (e.g. \( c_1 = \text{Network\_maxRate} \)); \( lv_b(1 \leq b \leq k) \) is a linguistic value (e.g. \( lv_2 = \text{high} \)); \( \text{value}_of(c_a, t) \) represents the value of context \( c_a \) at time \( t \); \( \mu_{c_a,lv_b}(x) \in [0, 1] \) is the pre-defined membership function of “\( c_a \) is \( lv_b \)”, which indicates when \( c_a = x \) to what degree \( c_a \) is \( lv_b \).

Given a service \( S(S_i \in S_{\text{need}}); \) each policy \( p_i \) in \( P \), is associated with a context situation that is most suitable for \( p_i \). Here, “most suitable” we mean the best balance of the tradeoff between resource consumption and QoS. Such a best-suitable context situation is referred to as the Standard Reference (SR) for \( p_i \) (denoted by \( SR(p_i) \)). If the actual context situation is better than \( SR(p_i) \), (e.g. actual Network\_maxRate is higher than the defined value in \( SR(p_i) \)), then there is waste of resource if \( p_i \) is used to deliver the service; similarly if the actual context situation is worse than \( SR(p_i) \), using \( p_i \) will not obtain the expected QoS.

**Definition 5 (SR(p_i)):** Given a set of linguistic values \( LV = \{Lv_1, Lv_2, Lv_3, \ldots, Lv_k\} \), \( SR(p_i) \) can be represented by a set of 3-element tuples:

\[
SR(p_i) = \{(c_a, lv_b, \mu_{c_a,lv_b}, \text{value}_of(c_a, t))|c_a \in C, a \in [1, n], lv_b \in LV, b \in [1, k]\}
\] (2)
Eq. (2) is almost the same as Eq. (1) except that the function value_of($c_a$, $t$) is replaced by best_value_of($c_a$, $t$) which represents the most suitable value of context $c_a$ when we use policy $p^i_j$ to deliver service. For a given set $P_i$, we call the aggregation \{SR(p^i_1), SR(p^i_2), SR(p^i_3), \ldots, SR(p^i_m)\}, $p^i_j \in P_i$, as the Standard Reference Depository of $P_i$ and denote it by $SRD(P_i)$.

**Definition 6 (Fuzzy Functional Adaptation Process):** FFAP is a mapping process from the current context situation $SI$(current) to a set of suitable policies $P_{suitable}$, where each element of $P_{suitable}$ is the most suitable policy for a certain service $S_i \in S_{need}$, the number of elements in $P_{suitable}$ is equal to the number of elements in $S_{need}$. Now we are ready to define the fitness functions. Although the aim of FFAP is to obtain $P_{suitable}$, the selection processes for elements in $P_{suitable}$ are similar and irrelevant from each other, thus the key point of FFAP is how to select the most suitable policy for a given service with making the best use of current resource and enhancing the user’s satisfaction. In practice, a context situation may not be exactly matched with any $SR(p^i_j)$. In order to select the most suitable policy from $P_i$, we should use proper fitness function to evaluate all the policies in $P_i$, so as to make the best choice.

**Definition 7 (Fitness Function):** Let $FD(p^i_j)$ be the fitness degree for policy $p^i_j$ under current context situation. Given a service $S_i \in S_{need}$, the fitness function $(FF)$: $SI$(current) $\times$ $SR(p^i_j)$ $\rightarrow$ $FD(p^i_j)$, is a mapping from the current context situation and standard reference $p^i_j$ to the fitness degree of policy $p^i_j$.

Here, we propose three different fitness functions based on three known distances: Manhattan distance, Minkowsky distance and Chebychev distance.

Manhattan $\_FF(SI$(current)$, SR(p^i_j)) = \frac{1}{\sum_{i=1}^{\text{size of } (SR(p^i_j))} |\mu(\text{best\_value\_of}(c_i)) - \mu(\text{value\_of}(c_i, \text{current}))|}$

Minkowsky $\_ FF(SI$(current)$, SR(p^i_j)) = \frac{1}{(\sum_{i=1}^{\text{size of } (SR(p^i_j))} |\mu(\text{best\_value\_of}(c_i)) - \mu(\text{value\_of}(c_i, \text{current}))|^p)^{1/p}}$

Chebychev $\_FF(SI$(current)$, SR(p^i_j)) = \frac{1}{\max_{i=1}^{\text{size of } (SR(p^i_j))} |\mu(\text{best\_value\_of}(c_i)) - \mu(\text{value\_of}(c_i, \text{current}))|}$

Where, size_of ($SR(p^i_j)$) represents the number of tuples in $SR(p^i_j)$, $\mu(x)$ is the membership function appears in $i$th vector, $p$ and $r$ are natural numbers.

In Minowky $\_ FF$ we can play independently on the two powers in the equation to find the balance between the large number of different elements and the importance of the difference between the two functions $(\mu_{value\_of} \mu_{value\_of})$.

The concept of fitness function is inspired by the membership function in classical fuzzy logic theory. But the two are different, in that the value of fitness degree is a positive number but not limited into $[0, 1]$, and the sum of fitness degree for all the policies for one service is not 1. The value of fitness degree only indicates that to what degree one policy is suitable for the current environment. In the above three functions, the denominators are for the calculation of the distance between $SI$(current) and $SR(P^i_j)$. After
obtaining the fuzzy distance, we calculate its reciprocal to get the fitness degree, i.e. when the distance between $SI$(current) and $SR(P^i)$ is 0, which means $SI$(current) and $SR(P^i)$ are completely consistent, then the fitness degree is infinity. Otherwise, the fitness degree will decrease with the increment of the distance.

6. AI PLANNING SERVICE COMPOSITION

We use Artificial Intelligent planning for our context-aware service composition framework, planning is a problem solving technique, where knowledge about available actions and their consequences is used to identify a sequence of actions, which, when applied in a given initial state, satisfy a desired goal. There are three main inputs to a planner: initial state, goal state and domain description. The initial state describes the starting state of the application domain, commonly called world. The goal state describes the desired world state. The domain describes actions that, when invoked, transform the world states. The output of the planning process is a plan, a sequence of actions that can be executed in order to achieve the desired goal state.

Fig. 5. Overview of the proposed service composition architecture.

Fig. 5 above shows an overview of the system architecture, which employs a process approach to service composition, to fulfill the design requirements outlined in the previous section. The four layers in the system architecture map to the four main stages in the service composition process. The first layer is the composition request management layer, which assembles and, if necessary, modifies a composition request. Each composition request is a formal definition of the user’s task intention. The next layer is the abstract service composition layer, which generates an abstract plan. An abstract plan is a set of abstract services and their control flow, comprising the composite service. Abstract services are high-level descriptions of service operations and cannot be directly invoked. The architecture specific service composition layer instantiates the abstract plan and generates a deployable service description, which represents a service instance. The deployable service description is passed to the execution and monitoring layer, which invokes the specified service instance and monitors its execution.

6.1 Composition Request Management Layer

Fig. 6 shows the structure of the first layer of the composition framework. The com-
position request is an entry point to the composition process. It specifies the user’s task and consists of two parts. The first part is a description of the core user task, for example, Mohcine’s request in Case 1 of the usage scenario for directions to the nearest Lebanese restaurant, selected from the Goal Service (step 1). The second part contains contextual parameters. For example, if Mohcine is using an android smartphone and he is currently moving near the shop center that would specify the computing device and the location, such contextual parameters further customize the composition request. For instance, in this context, it may be more appropriate to read out the driving directions to Mohcine. This layer constructs the composition request and feeds it to the abstract service composition layer (step 3).

If the abstract service composition process fails (step 4 (a)) control is passed back to the composition request management layer, which attempts to transform the composition request into an alternative request that can be satisfied. For example, Mohcine’s original composition request to find the nearest Lebanese restaurant may be replaced by a more generic request of finding any type of restaurant nearby. Furthermore, the functionality to present the output in speech format may be added if the speech synthesizer service is available and Mohcine makes the kit.

6.2 Abstract Service Composition Layer

The service composition process is split into two stages: abstract and architecture specific. Abstract service composition is the process of assembling abstract services, which are generic operations each satisfying different parts of the overall composition request. Architecture specific composition layer instantiates these abstract services and constructs an executable composite service. This two-layered approach has been introduced for a number of reasons. Firstly, this approach enables the framework to be implemented using any type of composition methodology, component technology and runtime environment.

Secondly, it facilitates recovery from service discovery and service execution failures, by isolating the different stages in the composition process. Finally, it enhances the scalability of the framework, as abstract service composition is performed only on a subset of abstract services, rather than all available service instances. Fig. 7 shows how the abstract service composition generates an abstract plan, which defines the control flow of abstract services. Firstly, the Translator Module converts the composition request to a
problem definition, which is in the representation format supported by the composition methodology in use (step 1 in Fig. 7). The Abstract Service Repository stores and manages abstract service descriptions. In our usage scenario the sample abstract services provided include a restaurant directory service and a speech synthesizer service. Abstract services are semantically annotated; their descriptions contain the types of parameters they expect, as well as preconditions and expected post-conditions for their successful execution. Each abstract service also points to the files carrying the descriptions of the domain concepts used, such as a definition of restaurant in our usage scenario. The Translator Module converts the available abstract service descriptions and domain concepts from the Abstract Service Repository to generate the domain description, in the representation format supported by the composition methodology used (step 2 in Fig. 7).

The Composition Engine uses the problem definition (step 1) and the domain description (step 2) to generate the abstract plan (step 3 (a)), which consists of a list of abstract services to be executed, described in the composition language. It is then stored in the internal representation format which is independent of the composition methodology. Finally, the abstract plan is fed to the architecture specific service composition layer for instantiation. If the system fails to create an abstract plan (step 3 (b)) control is passed back to the composition request management layer, where the composition request is transformed into one that may also be satisfiable. If in the architecture specific composition layer (layer 3) the process of service discovery and instantiation fails, control is passed back to the abstract service composition layer, which initiates a recomposition process (step 4). The Composition Engine may be implemented by a number of different composition methodologies. For example, AI Planning has proven to be a valuable and effective tool for service composition [18] Abstract services can be represented in terms of their non-functional and functional properties. Non-functional properties describe service provider details and Quality of Service parameters. Functional properties contain descriptions of service operations in terms of inputs, outputs, preconditions and effects, which make it easy to convert them into planning actions. The Translation Module converts a composition request into problem definition and abstract service descriptions in the domain description, which are formats supported by the planner.

6.3 Architecture Specific Service Composition Layer

Fig. 8 shows the system components involved in the process of architecture specific service composition and their interactions. The Plan Translator (step 1 in Fig. 8) converts
the abstract plan into an abstract execution plan, which describes a composite service in architecture specific format. As the framework stores the abstract plan in an internal representation format, it is necessary to have translation mechanisms for different runtime technologies used and their corresponding representation formats. The abstract execution plan describes each service in terms of its parameters, expected preconditions and post conditions, and any other semantic tags such as service categorisation codes, as well as Quality of Service parameters.

The Plan Instantiator executes the abstract execution plan and mediates the process of service discovery and instantiation. The Service Registry allows service providers to submit descriptions including their identifiers, name, interfaces provided, and time-to-live information. It exports interfaces for service discovery and publishing. It performs service discovery and returns the service binding information for each service instance.

The Plan Instantiator processes the abstract execution plan and passes the information about abstract services and the required Quality of Service parameters to the Service Registry (step 2). For example, the abstract execution plan may contain an abstract service representing a restaurant directory. Following the discovery process, this abstract service may be instantiated by restaurant directory service. Once the abstract service is instantiated the Service Registry returns its service binding. Plan Instantiator uses this service binding as a basis for the deployable service description (step 3 (a)) and passes it to the execution and monitoring layer, which schedules its invocation. If service discovery fails (step 3 (b)) control is passed to the abstract service composition layer, which triggers recomposition. However, if the service fails during execution, control is passed back to architecture specific composition layer, where a replacement service is fetched (step 4).

6.4 Execution and Monitoring Layer

The Execution Engine provides the runtime environment in which services can be executed. It invokes scheduled services as specified in the deployable service description (step 1 in Fig. 9). The Monitoring Engine is bound to the Execution Engine to track changes in the runtime environment, service performance and composition request status. The Monitoring Engine verifies the service preconditions before being invoked by the Execution Engine. During the service lifetime it observes changes in the environment and propagates any failures to the upper layers in the framework, where they are dealt with. Finally, once the service completes its operation the Monitoring Engine verifies service effects, against the expected outcomes. Service execution may fail due to network disconnection. If a service instance cannot be invoked the system tries to execute a
replacement service, if one has been previously cached. Pointers to replacement services may be included in the deployable service description. If the cached service fails as well, control is passed to the architecture specific composition layer, which replaces it with a suitable service of the same type (step 3 (a)). If this operation fails too, the system continues propagating the failure up the layered framework structure. Should an unanticipated change in context occur (step 3 (b)), or should the user change the task specification (step 3 (c)), control is passed to the composition request management layer, where a new composition request is generated and recomposition triggered. The Monitoring Engine updates the state of the Composition Engine and the Execution Engine. There are several different events that may take place during the service execution. For example, the actual outcome of the service may not be as anticipated or the primary aim of the service may be unexpectedly satisfied by another service. For instance, the expected outcome of the scheduled service is to automatically lower the volume of the in-vehicle stereo. Before the service is executed, the user manually adjusts the stereo volume and therefore achieves the outcome of the scheduled service. In such cases the Monitoring Engine adds the information to the state description in the Execution Engine, which ensures, it does not trigger the service execution. Finally, if a required service precondition is no longer true the service will not be invoked. To observe context changes and service execution the Monitoring Engine employs monitoring procedures proposed by Haigh et al. [17].

![Fig. 9. Execution and monitoring layer.]

### 7. CONCLUSION

In this paper, we have proposed an approach for context-aware services adaptation. In our approach, the adaptation process takes into account three phases: (i) discovering phase based which result on susceptible services to meet the user’s context, (ii) policy-based adapting phase to rank-order the discovered services according to their policy to implementing the service. User context are modeled with fuzzy predicates and the rules for adopting policies are interpreted thanks to linguistic quantifiers, this makes the adaptation process more flexible, and (iii) service composition phase to combine some services together to form a new adaptable service.

The composition framework employs a layered design approach to separate its four stages and functions recursively to ensure a high QoS in the composition process. The framework successfully uses AI planning to control service composition based on the user’s context. Contextual changes may trigger recomposition of services during execution, causing the application to evolve dynamically.

In our future research, we will continue to enhance our approach in the categories of context modeling, service discovery, service adaptation, and service composition. We are
currently working on a prototype system to evaluate our approach by conducting more experiments to examine performance metrics including efficiency of service composition, effectiveness of service policy ranking, and reliability of context modeling.

REFERENCES

348-363.

Mohcine Madkour is Ph.D. candidate at the Department of Computer Science of University Mohameed 5 Agdal of Rabat, Morocco. His research interests include the pervasive systems and intelligent spaces ontology, context-awareness, service adaptation and retrieval. He has some published articles and lot of participations in international conferences.

Driss El Ghanami got his Ph.D. degree in robotics science from the University of Versailles-France. He is a full Professor and a senior researcher in Computer Science at the Mohammadia School of Engineering of the University Mohamed 5 Agdal, Morocco. His research focuses on wireless network and embedded information systems.

Abdelilah Maach got his Ph.D. degree in computer science from the University of Ottawa. He obtained “Habilitation à Diriger les Recherches” from the University Mohamed 5 of Rabat Morocco. He is a full Professor in Computer Sciences and leader of the Network and Information Systems Group at Mohamed 5 University.