Context-aware Dynamic Service Composition in Ubiquitous Environment

K. Tari, Y. Amirat, A. Chibani, A. Yachir and A. Mellouk,

Abstract—**The service composition aims to provide a variety of high level services. Recent approaches cannot fully satisfy the requirement raised by ubiquitous environment. In this paper, we propose a layered design framework which aims at being flexible and robust to failure service composition. It adopts an abstract way of generating plan using rule-based techniques in order to adapt to the changes occurring on the services and the context of use. The approach optimizes the number of services and the recomposition time in large-scale environment by removing the phase of rediscovery. The framework for service composition and monitoring includes learning mechanism for the service selection, based on an estimation of the reputation for abstract services and the quality (QoS) for concrete services. The proposed approach is tested, under USARSim simulator, on a set of ubiquitous services for assisting elderly or dependant person in a residential environment. The obtained results show the feasibility and the scalability of the approach and a better reactivity to the dynamic and uncertain nature of the ubiquitous environment.**

Index Terms— **Dynamic service composition, QoS-Based Learning mechanism, Context awareness, ubiquitous environment**

I. INTRODUCTION

With the fast emergence of ambient intelligence, ubiquitous computing, sensor networks and wireless technologies, service composition is becoming an active research domain of pervasive computing. The ubiquitous environment is closely related to the long term vision of an intelligent service system in which technologies are able to automate a platform embedding the required devices for providing context aware, personalized, adaptive and anticipatory services.

Many works have been conducted in our laboratory [1] to deal with ubiquity issues. The purpose of our research is to integrate seamlessly smart objects located in a ubiquitous space (mobiles handsets, sensors/actuators, robots, etc.). Smart objects interfaces are implemented using OWL-S standard language [2]. In this context, Chibani et al. [1] have proposed a Context-Aware Middleware for supplying intelligent services to users. The work presented in this paper is in the continuity of these works.

In ubiquitous ambient environment, simple and complex services are provided to users, according to their contexts, at anytime, anywhere, and using any available device. On the other hand, the dynamic and uncertain nature of this environment is due to the frequent changes that occur in it. These changes occur in different situations: a new device offering services with better quality is detected, a service in use fails or its quality declines, changes in user needs, loss of services due to the mobility of the user or the devices, etc. In this sense, ubiquitous environment requires composite services able to adapt dynamically to these frequent changes. In order to support this, the management system needs to adapt dynamically the composite services, even during their execution, by selecting appropriate services according to a composition plan, to ensure an automatic and optimal execution of the services.

In [3], the authors studied and compared dynamic service composition approaches for ubiquitous environment. These approaches cannot fully satisfy the requirement raised by ubiquitous environment. Hence, this paper is focused on the dynamic service composition in the ubiquitous environment.

The service composition consists of creating complex services by combining existing atomic services in order to take into account user and context of environment. For this purpose, the service composer defines an appropriate action plan to perform a composition task and then determines the most appropriate service for each action of the plan.

On the basis of our previous work [1] and in order to address these concerns, we have opted to change the way of composing services. Indeed, for a better recomposition in case of failure, we have removed the phase of rediscovery to save time. This phase is particularly time-consuming in large-scale environment. In this paper, we proposed a layered design architecture for flexible and failure-tolerant service composition. Thus, we presented the planning algorithms and monitoring mechanism for dynamic service composition. This framework includes a learning mechanism for selecting services. It allows dealing more efficiently with the dynamic and uncertain nature of the environment.

The remainder of the paper is organized as follows. Section II describes the proposed architecture for dynamic service composition. Section III describes a use case with a detailed scenario for assisting an elderly person. Section IV explains the implementation and tests results. Section V gives some related work. Finally, section VI gives the conclusion and future work.

Manuscript received September 28, 2009. This work was supported in part by the LISSI Laboratory of Paris12 University (France).

K. Tari, Y. Amirat, A. Chibani and A. Mellouk are with Laboratory of Image, Signal and Intelligent Systems (LISSI), University of Paris Est-Paris 12 Val de Marne, France (e-mails: {karim.tari, amirat, chibani, mellouk} @univ-paris12.fr).

A. Yachir is with Department of Computer Science, Military Polytechnic School (EMP), Bordj El Bahri, Algiers, Algeria (e-mail: a_yachir@yahoo.fr).

II. DYNAMIC SERVICE COMPOSITION ARCHITECTURE

In this paper, we distinguish two types of services, as it is generally considered in the literature [5], *abstract* and *concrete* services. Such an approach has the advantage of de-coupling higher level service descriptions from concrete message specifications and consequently allows to layer more easily on top of industry adopted standards. According to the standard specification of a service profile – advertises the service to external entities (Owl-S profile description [2]) – and the specification adopted in our previous works [1], we define a concrete service *i*, noted cs_i, as a service that performs some functionality by acting on input data (cs_i^{in}) in order to produce output data (cs_i^{out}). It is described by the following set: $(cs_i^{in}, cs_i^{out}, Prec, Eff, Q\text{-}value)$. Where The conditions on the inputs are called preconditions (*Prec*), the values of the outputs are called effects (*Eff*) and *Q-value* $\{QoS_1, QoS_2, \ldots, QoS_k\}$ are defined as a set of *Qualities of Service* (*QoS*) (set of quality requirements of a service). We extended also the definition of an abstract service, given in [1]. An abstract service, noted AS_i, is a task that can be achieved by several concrete services that offer the same functionality and possess the same input parameters and subset of output parameters represented as the output parameters of AS_i. It is described by the following set: $(AS_i^{in}, AS_i^{out}, R, AS_i^{CS})$ such as: $AS_i^{CS} = \{cs_{i,1}, cs_{i,2}, \ldots, cs_{i,n}\}$, R represents the reputation of the AS_i and $(\forall\ cs_{i,j}, cs_{i,k} \in AS_i^{CS} / (AS_i^{in} = cs_{i,j}^{in} \cap cs_{i,k}^{in}) \wedge (AS_i^{out} = cs_{i,j}^{out} \cap cs_{i,k}^{out})$.

We assume the hypotheses in [1], and the following hypothesis: Each user has its own profile, which is composed of: user status, user personal information, user preferences, user security preferences and mainly the minimum threshold *QT* of the global quality of a service that the user can accept in addition to the weight of each quality of service according to its importance for the user.

We consider three types of composition plan: the template plan and the optimal plan which are abstract plans, and the execution plan which is a concrete plan instantiated from the optimal plan.

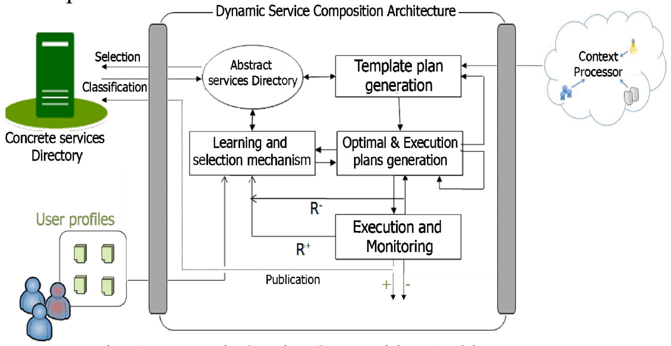

Dynamic Service Composition Architecture
Selection
Classification
Concrete services Directory
User profiles
Abstract services Directory
Template plan generation
Learning and selection mechanism
Optimal & Execution plans generation
Execution and Monitoring
R-
R+
Publication
+
-
Context Processor

Fig. 1. Dynamic Service Composition Architecture

Our service composition approach *CDSC* (*C*ontext-aware *D*ynamic *S*ervice *C*omposition) is a layered design approach for flexible and failure-tolerant service composition [6] (Fig1). More precisely, the architecture we used includes three communicating layers. The first layer consists in the automatic generation of a template plan, using rule-based techniques. The plan contains all possible abstract services–that could contribute to compose the requested service. This plan is used, in the second layer, on the one hand for the generation of an optimal abstract composition plan and on the other hand, for the regeneration of a new optimal plan in case of failure (Fig.2). The optimal plan is selected according to the reputation of the abstract services and complementarity of their output parameters. After the generation of the optimal plan, the concrete service with the best quality is selected for each abstract service instantiating an execution plan. In order to take into account the dynamic and the uncertain nature of the ubiquitous environment, a learning mechanism is implemented to update the QoS of each invoked concrete service as well as the reputation of the corresponding abstract service. In the proposed approach, we distinguish two types of quality the QoS and the QoE. From the learning model, QoS attributes are updated according to the performance of each concrete service. The QoE (Quality of user Experience) is a metric representing the overall quality of service as it is finally perceived by the end-user with respect its needs and preferences.

The third layer consists of monitoring the *execution plan* and the automatic adaptation of the latter in case of decrease of the quality of a concrete service, or failure of the latter. The adaptation of the plan allows the replacement of this concrete service by another one (best quality) belonging to the same abstract service class. In certain situations, this replacement may become inoperant because no concrete service has a sufficient quality. The optimal plan is then locally updated using the template plan to replace the corresponding abstract service. Unlike in [1], this local recomposition avoids additional treatments: rediscovery of services and regeneration of a new composition plan. In case of successful execution of composition plan, the composite service is published for further use within in the *concrete services directory (CSD)*.

In the remainder of the paper, we present the different concepts and approaches used in this algorithm.

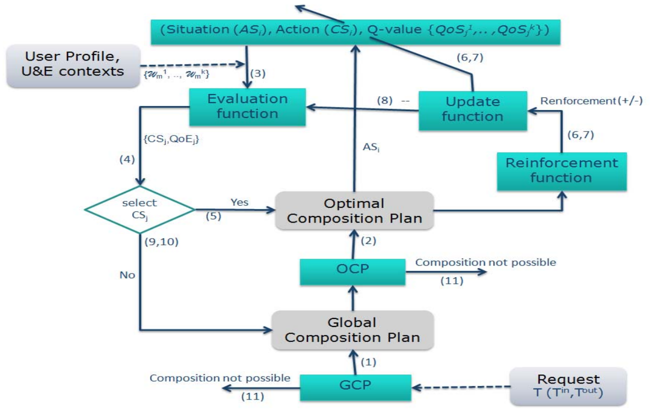


Fig. 2. Composition and Monitoring of services.

A. *Composition Plan Generation*

As in [4], in the proposed approach, a composition plan is expressed as a graph that contains all services that compose the request T from the goal T^{out} (requested output parameters) to retrieve the T^{in} (input parameters). The generation of composition plan becomes then a planning graph problem. This graph represents an inverse service composition plan. This graph G is a pair (V, E) of sets V and E, where V is a set of vertices (sets of required parameters) in the graph, and E is a set of edges represented by a set of quadruples (RP_i, RP_j, AS_j, R_j), where RP_i, $RP_j \in$ V.

A.1. Template Plan Generation

The generation of a template plan consists of the automatic construction of an abstract composition plan, using rule-based techniques. The plan contains all possible abstract services that could compose the required service. Initially, the graph G = (V,E)=($\{RP_0\},\phi$). For each requested parameters RP_i, abstract services are selected from the *abstract services directory (ASD).* These selected abstract services, called *Abstract Services Candidates,* have at least one output parameter belonging to RP_i. If all the data provided by these abstract services candidates include the requested parameters RP_i, then all the requested input parameters of the abstract services candidates are added to V and all the corresponding edges (RP_i,RP_k,AS_k,R_k) are added to E; otherwise, the RP_i is deleted from the graph because no solution exists. The generation of the graph G stops when all the requested parameters are identified or when the composition is impossible.

A.2. Optimal Plan Generation

Once the template plan is obtained, an optimal abstract composition plan is selected according to the reputation and the complementarity of the parameters provided by the abstract services candidates. This selection avoids the invocation of services that provide the same parameters. Therefore, the generated optimal plan has the lowest number of services and parameters that can appear in the composition. This automatic selection – done from the top of the graph to its end-vertices using the Level-order Tree Traversal Method [7] – allows us to avoid also the rediscovery phase in case of failure by reselecting locally the new optimal plan. The optimal plan generation is based on rules and ranking techniques. As seen before, the graph is an inverse plan of service composition plan. Then once the optimal plan is derived, its execution is done by traversing a graph from its end-vertices to its top. All the abstract services located in the end-vertices of the graph start at the same time and are executed in parallel.

B. Ranking-based Service Selection

After the generation of the *optimal plan*, a concrete service is selected for each abstract service generating then the *execution plan*. The concrete services selected have the best quality of Experience (QoE) superior than then the threshold specified by the user. This selection is expressed as follows:

```
For each $OAS_i$:
    For each $\underline{AS_j} \in OAS_i$:   Replace $AS_j$ by $CS_k$ /
        $CS_k = ArgMax(QoE_t(x))_{x \in AS_j^{CS}} \wedge CS_k > QT$
```

Where: - AS_j^{CS}: represents the set of concrete services belonging to the abstract service class AS_j.

- $QoE_t(x)$: represents quality of a concrete service x at time t.
- QT: represents the minimum threshold of QoE that the user can accept.
- *ArgMax:* return the concrete service for which the value of the *QoE* attains its maximum value.

B.1. QoS and QoE Estimation

The quality of concrete services (QoS) depends on the quality of their smart objects (applications, sensors/actuators, robots, devices, etc.) located in a ubiquitous space. Therefore, the quality parameters of a service – such as the response time, the availability, reliability, lifetime (energy, expiration date), scalability, confidentiality, security, cost, etc. – are objectively measured. On the other hand, we work on services at application layer taking into account the user and environment contexts (elderly or dependant person needs, night, gas-leakage, fire, etc.). Therefore, we use the Quality of Experience (QoE) [8], known as "Quality of User Experience", which is a subjective measure of user's experience with corresponding service. This definition includes the user and the environment in the assessment, and demands an appropriate weighting of objective quality parameters (QoS). Therefore, the context processor associates a weight to each QoS parameter according to the desired user preferences (specified in the user profile), and the user and environment contexts. The user expresses her/his special needs and requirements in the preferences of her/his profile through the weights of QoS parameters. For example, to select the fastest services on detection of a gas leak (environment's context) or a malaise of an elderly person (user context), the weight of response time parameter should be important.

$QoE_t(s)$: Quality of User Experience obtained when achieving the service *s*. It is estimated using the following formula:

$$QoE_t(s) = \sum_{i=1}^{NbQ} (W_k^i \times QoS_t^i(s)) \qquad W_k^i = \frac{\left(WUP_k^i + WUC_k^i + WEC_k^i\right)}{N}$$

Where: - $QoS_t^i(s)$: quality *i* of the concrete service *s* at the time t.

-*NbQ*: number of quality parameters of service.

- W_k^i: weight of the quality *i* for the user k.
- WUP_k^i: weight of the quality *i* for the user k according to its preferences (specified in the user profile).
- WUC_k^i: weight of the quality *i* for the user k according to its context.
- WEC_k^i: weight of the quality *i* for the user k according to the environment context.

-*N:* number of collected information (WUP_k^i, WUC_k^i, WEC_k^i).

B.2. Reputation Estimation

The reputation of an abstract service depends on the quality of its concrete services. More precisely, it depends on the success or not of the abstract service when it is selected. It is estimated by the following formula:

$$R_t(a) = \frac{Nb_{\text{success}}}{Nb_{selection}}$$

Where: $R_t(a)$: represents the reputation of the abstract service *a* at the time t.

If the selection of the abstract service *a* is successful then its reputation increases. Otherwise, if the abstract service does not find a valid concrete service or the *CS* selected is not successful, then its reputation decreases.

C. Monitoring and QoS-based Learning Mechanism

C.1. Monitoring

Once the optimal plan is generated and the concrete services are selected, the monitoring of the execution plan is done to guarantee an automatic adaptation and a failure-tolerant. To avoid re-invocation of concrete services belonging to the same abstract service, the following rule must be respected at each set of *Optimal Abstract Services (OAS$_i$)*:

If $((AS_i^{out} \cap RP_i) \subseteq AP_t)$ **then** switch to the next AS_i. **End If;**

Where: AP_t: represents all the parameters available at time t (the input parameters of the request T (T^{in}) and the parameters obtained during the execution of the *execution plan*).

C.2. QoS-based Learning Mechanism

Due to the mobility of users and services, the nondeterministic behavior of the latters and the uncertain nature of ubiquitous environments, the composition plan must be flexible and failure-tolerant. For this purpose, the proposed approach integrates a learning mechanism on the *QoS* of concrete service and on the *Reputation* of abstract service. All quality parameters of all concrete services and all reputation of all abstract service are equally initialized.

$\forall i \in NbQ, \forall s \in CSD : QoS_0^i(\text{s}) = Q$ and $\forall a \in ASD : R_0(\text{a}) = R$.

According to the ending of invoked service, the learning mechanism rewards (positively or negatively) the concrete service by calculating its new quality parameters and estimates the new reputation of its abstract service.

C.2.1. QoS Monitoring

The proposed architecture uses Q-learning as reinforcement-learning technique because it provides good experimental results in terms of learning speed [9]. The reinforcement-learning module can be quite reactive to decide which next action to take. However, the learning speed depends on the complexity of the task. In our case, the task consists of analyzing the responses obtained after the invocation of concrete services to update their quality parameters.

The problem model consists of an agent, states *AS* and a number of actions *(CS$_i$)* per state *(AS$_j$)*. By performing an action CS_i, where $CS_i \in AS_j^{CS}$, the agent can move from state to state. Each state provides the agent a reward or punishment (a negative reward). More precisely, for each situation *(AS$_i$)*, the learning module determines the action *(CS$_i$)* that maximizes the expected Q-value $\{QoS_j^1, \ldots, QoS_j^k\}$. The quality parameters [10] of the concrete service *s* (Fig. 2) are updated in the following manner:

TABLE I: TABLE OF QUALITIES OF CONCRETE SERVICES

Quality	Quality of Concrete service (QoS)	Weight user i
Response Time (RT)	$\frac{1}{TR}$	W_i^1
Availability (A)	$\frac{\text{Nb}_{\text{Available}}}{\text{Nb}_{\text{Requested}}}$	W_i^2
Reliability (Re)	$\frac{\text{Nb}_{\text{DataReceived}}}{\text{Nb}_{\text{DataPromised}}}$	W_i^3
Security (S)	Security level	W_i^4

According to the characteristics of the ubiquitous environment (seen in II.*C.2*), we take into consideration the following quality parameters for concrete services: Response Time, Availability and Reliability. In our learning model, more the interval of time (Δt) between two invocations of a concrete service *s* is large, more importance is given to the last quality estimation (the reverse is also true). Also, If a concrete service is not available or it has missing output data (reliability) then the reinforcement is negative (Fig.3.(8)) else the reinforcement is positive.

- *Response Time:* represents the quality of the time required for a concrete service *s* to respond to a given input.

$$RT_{t+1} = \frac{t * RT_t + \Delta t * RT_A}{t+1}$$

- *Reliability:* quantifies the reliability of *s* in terms of received data.

$$\text{Re}_{t+1} = \frac{t * \text{Re}_t + \Delta t * \text{R}e_A}{t+1}$$

- *Availability:* quantifies the availability of *s*.

$$A_{t+1} = \frac{Nb_{Available_{t+1}}}{Nb_{\text{Re}quested_{t+1}}}$$

Where: $-\Delta t = \lceil (t+1) - t \rceil$: represents the interval of time between the two last invocations of a concrete service s.
- *TR*: represents the response time in second, calculated from the invocation of the concrete service.
- RT_t, RT_{t+1}, RT_A : represent respectively the old, the future and the current quality of the response time. Response time is inversely proportional to its quality (Table. I).
- Re_t, Re_{t+1}, Re_A : represent respectively the old, the future and the current quality of the reliability (Table. I).

In the proposed approach, we took into account the three previous quality parameters by default. For every quality, a specific learning rule is applied. Thus, for each new added quality, a rule must be defined.

III. USE CASE

The proposed approach is tested on a set of ubiquitous services which provide comfort, monitoring and assistance for elderly or dependant person in residential environment. The scenario we have studied and simulated under USARSim environment is described as follows: an external supervisor, such as doctor or family member, needs to check the state of his patient located in a residential environment, broadcast him alert or information and make sure that the patient received the information.

A. ENVIRONMENT DESCRIPTION

The test scenario is implemented in Java and integrated in the USARSim environment (Fig. 3). The latter is a high fidelity simulation of urban search and rescue robots and environments intended as a research tool for the study of human-robot interaction (HRI) and multi-robot coordination [11]. The robot used in this scenario is a P2AT. It is equipped with a PTZ camera, INU (Inertial Navigation Unit) Sensor, odometry sensor, RFID Sensor (1 RFID Reader on the P2AT robot, 1 RFID Tag on the person), touch screen, Human Motion Sensor, Sound Sensor and 16 Sonar sensors equally distributed around it, enabling the robot to sense obstacles in all directions [1]. The test scenario implies also a wheelchair, a set of smart display devices (touch screen, tv) with a VoIp/VideoIp systems.

TABLE II: TABLE OF ABSTRACT SERVICES

Abstract Services Description	Input	Output
Robot_State (AS_1)	a	f
Human_Motion_Analysis (AS_2)	d	h
Human_Sound_Analysis (AS_3)	e	i, j
Trajectory_Computing(AS_4)	b, f, g	k
Human_State_Analysis (AS_5)	h, i, j	m
Robot_Control(AS_6)	k	n
StartSmartDisplay (AS_7)	n	n
LocateSmartDisplay (AS_8)	c	g
InfoPerson (AS_9)	g	P
Sensor_Person_Location (AS_{10})	c	G
Wheelchair_Control (AS_{11})	k	N
Wheelchair_State(AS_{12})	q	f, n

B. SCENARIO DESCRIPTION

The service composition system receives all the data (a, b, c, d, e, q) collected from the external supervisor and from the corresponding sensors (Table II). The goal of the system is to locate and check on the status of the patient in the first place and transmit then external information or alerts adequately according to the context of use. For example, if the patient has an inability to move then the system uses the wheelchair to approach him to the nearest smart display device. In case of, the system brings to him the robot with a smart display device.

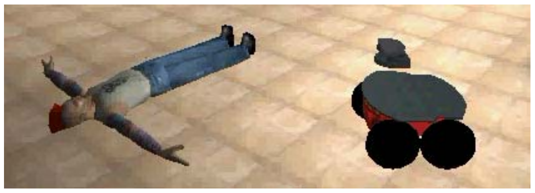


Fig.3. P2AT the robot moves to assist the person in USARSim environment.

C. SERVICE COMPOSITION AND MONITORING

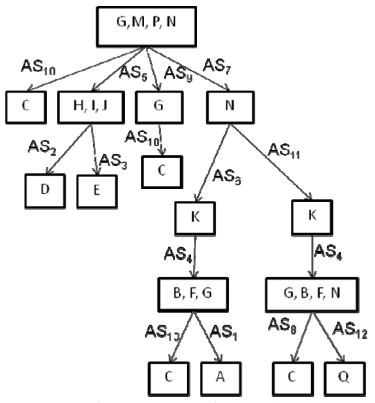


Fig.4. Template plan

To process the request T, such as: T^{in}={a, b, c, d, e, q} and T^{out}={g, m, p, n}, the service composition system generates the template plan (Fig.4). The optimal plan (Fig.5) is generated according to the context of use. In our case, the patient has an inability to move then the system uses the wheelchair to approach him to the nearest smart display device.

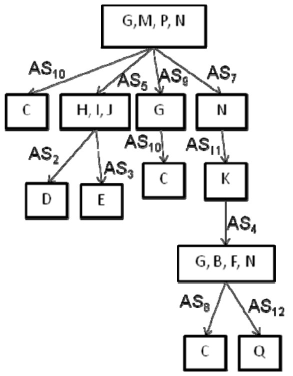


Fig.5. Optimal plan

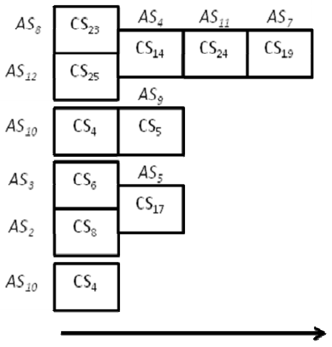


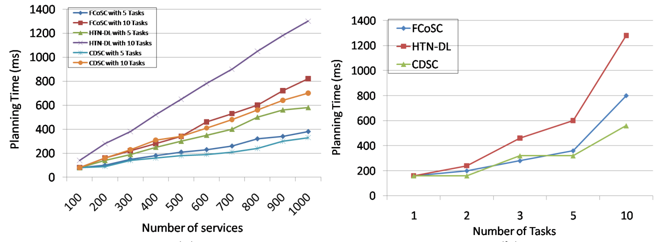
Fig.6. Execution plan

For each abstract service (Fig.6), a concrete one, with the best QoS, is selected. In case of failure of a concrete service (*CS_{19}*), the system replaces it locally by the best concrete service belonging to the same *AS_7* (*CS_{20}*). In case of failure of an abstract service *AS_{11}* (*Wheelchair_Control*), the system regenerates a new optimal plan. It brings to the user, the robot (*AS_6*) with smart display device.

This use case shows the feasibility of our approach in simulated ubiquitous environment.

IV. IMPLEMENTATION AND TESTS

We have measured the evolution of planning time of our currently proposed algorithm as a function of number of services. Then we have compared the performance of our algorithm with our previously proposed algorithm FCoSC[1] and HTN-DL algorithm[12]. All experiments were performed on a PC with the Pentium 4 with 256Mo RAM, running Windows XP at 1.6GHz. Firstly, we have fixed the number of tasks to 5 and 10 and varied the number of tasks from 100 to 1000 (Fig. 7.a). Secondly, we have fixed the number of services to 1000 and varied the number of tasks from 1 to 10 (Fig. 7.b).



(a) (b)

Fig. 7. Planning time *vs.* the number of services/tasks.

Our proposed algorithm *CDSC is* more time-efficient as compared to *FCoSC* and *HTN-DL* (Fig. 7). The results show a better scalability and reactivity to the dynamic and uncertain nature of the ubiquitous environment of the proposed approach. This better performance is achieved by i) generating abstract plan instead of concrete plan in *HTN-DL*, ii) removing the phase of rediscovery by using template plan and iii) generating the plan from the requested data instead of initial input [1].

V. RELATED WORK

In the literature, several approaches have been proposed for dynamic service composition but without a sufficient attention to the ubiquitous environments. The HTN planning [12] is described as an AI planning methodology that creates plan by task decomposition. The planning system decomposes the tasks into smaller subtasks until primitive tasks are found that can be executed directly. In [13], Doshi et al propose a workflow based approach that interleaves MDP-based workflow generation and Bayesian model learning to model the stochastic behaviour of the Web services. In [14], the authors propose a Context-aware Dynamic Service Composition architecture using the SHOP2 planning system for the selection of required services and the sequence of their execution and the BPEL4WS Web service composition technology. A workflow-based composition language for Web service based on XML, is proposed to express the logic of the composite service. Nahrstedt et al. [15] exploit the idea of the QoS-based global planning algorithms for dynamic service composition. The generated plan is revised, if necessary, during the execution. In [16], the authors propose a service composition frameworks based on multi-agent systems *(MAS)* and context-aware systems to provide the user with the best services in a pervasive environment. The use of Agent technology offer interesting advantages for helping users, discovering and composing services.

VI. CONCLUSION

In this paper, we have presented a layered design approach for flexible and failure-tolerant service composition. It generates automatically flexible plans, and optimizes both the number of services and the parameters that appear in the composition. In order to fit to the changes occurring on the services and to the context of use, the plan is generated in an abstract way, using rule-based techniques. This, allows us also to optimize the time of the recomposition in large-scale environment by removing the phase of rediscovery. The framework for service composition and monitoring includes also learning mechanism for the service selection, based, on one hand, on estimation of the reputation of abstract services, and on the other hand, on the quality (QoS) of concrete services. The QoE estimation allowed us to take in consideration the user's preference and the environment context in the service selection. The obtained results show the feasibility and the scalability of the proposed approach and a better reactivity to the dynamic and uncertain nature of the ubiquitous environment. In future works, we shall introduce on the one hand semantic technology to manage the context and on the other hand policies to deal with security issues.

REFERENCES

[1] A. Yachir, K. Tari, A. Chibani and Y. Amirat, "Towards an Automatic Approach for Ubiquitous Robotic Services Composition ". In Proc. Of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2008), Pp.3717-3724, Nice, France, 2008.

[2] The OWL Services Coalition. "OWL-S: Semantic Markup for Web Services" (http://www.daml.org/services/owl-s/1.0/owl-s.html), 2003.

[3] A. Urbieta , G. Barrutieta , J. Parra , A. Uribarren, "A survey of dynamic service composition approaches for ambient systems", Proceedings of the Ambi-Sys workshop on Software Organisation and MonIToring of Ambient Systems, p.1-8, Canada, February 2008.

[4] S. Oh, D. Lee, and R.T. Kumara Soundrar. "A Comparative Illustration of AI Planning-based Web Services Composition". Penn State University. ACM SIGecom Exchanges, Vol. 5, No. 5, December 2005.

[5] S. Balzer,T. Liebig,"Bridging the Gap Between Abstract and Concrete Services A Semantic Approach for Grounding OWL-S", Proceedings of the Workshop on Semantic Web Services:Preparing to Meet the World of Business Applications, November 2004, Hiroshima, Japan.

[6] J.Y. Tigli, M. Riveill, G. Rey, S. Lavirotte, V. Hourdin, D. Cheung, E. Callegari, "WComp Middleware for Ubiquitous Computing: Aspects and Composite Event-based Web Services", in Annals of Telecommunications, January 2009.

[7] Tree Traversal Methods, http://www.macs.hw.ac.uk/flex/BScCS/ds1/activity10.html

[8] B. Corrie, H. Wong, T. Zimmerman, S. Marsh, A.S. Patrick, J. Singer, B. Emond, S. Noël, "Towards quality of experience in advanced collaborative environments", the 3rd Annual Workshop on Advanced Collaborative Environments, Seattle, Washington, June 22, 2003.

[9] S. Gadanho, "Reinforcement Learning in Autonomous Robot : An Empirical Investigation of the Role of Emotion", Ph.D. dissertation, University of Edinburgh, 1999.

[10] F.M. Huang, C.W. Lan, S.J.H. Yang, "QoS-Based Learning Services Composition for Ubiquitous Learning", IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (STUC 2008), pp.513-518, 2008.

[11] J. Wang, S. Balakirsky. "USARSim Manual V3.1.1: A Game-based Simulation of mobile robots". University of Pittsburgh, National Institute of Standards (NIST), USA. 2007.

[12] E. Sirin, B. Parsia, and J. Hendler. "Template-based Composition of Semantic Web Services". MINDSWAP Research Group, University of Maryland, College Park MD 20742, USA. In American Association for Artificial Intelligence (www.aaai.org), 2005.

[13] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma. "Dynamic Workflow Composition using Markov Decision Processes". Int. Journal of Web Services Research, 2(1): 1-17, Jan-March 2005.

[14] M. Vukovic, P. Robinson, "Adaptive, planning-based, Web service composition for context awareness", International Conference on Pervasive Computing, Vienna, 2004.

[15] K. Nahrstedt, D. Xu, D. Wichadakul, and B.Li. "Qos-aware middleware for ubiquitous and heterogeneous environments". IEEE Communications magazine, 39(11):2–10, 2001.

[16] Z. Maamar, S. K. Mostefaoui, H. Yahyaoui. "Toward an agent-based and context-oriented approach for web services composition". IEEE Trans. on Knowledge and Data Engineering, 17(5):686–697, 2005.