Ubiquitous Cloud: Managing Service Resources for Adaptive Ubiquitous Computing

Koichi Egami, Shinsuke Matsumoto and Masahide Nakamura
Graduate School of System Informatics, Kobe University
1-1 Rokkodai-cho, Nada-ku, Kobe, Hyogo 657-8501, Japan
Email: egami@ws.cs.kobe-u.ac.jp, {shinsuke, masa-n}@cs.kobe-u.ac.jp

***Abstract*—The *adaptive ubiquitous services*, which dynamically adapt behaviors to requirements and contexts, are one of the major challenges in the ubiquitous computing. To facilitate the management of ubiquitous service resources, this paper presents a novel platform called *ubiquitous cloud*, borrowing the concept of the cloud computing. The ubiquitous cloud supports various stakeholders to use appropriate ubiquitous objects in infrastructure, platform and application levels. We present an architecture consisting of four key components: the service resource registry, the adaptive resource finder, the context manager and the service concierge. To demonstrate the effectiveness, we develop a practical adaptive service "room environment relocation service" in an actual home network system, with and without the proposed ubiquitous cloud.**

***Keywords*-adaptive ubiquitous services, cloud computing, service-oriented architecture, service resource discovery**

I. Introduction

Along with the diversification of social needs, ubiquitous services and applications are required to be more *adaptive* to users and surrounding environments. In the ubiquitous and pervasive communities, many different techniques have been proposed to achieve the adaptive services including, changing the composition, re-assembling workflows, optimizing behaviors, performing AI planning (e.g., [2] [3] [5]). In reality, however, it was quite difficult to implement these theories in practical applications, since there was no standard platform to guarantee the interoperability among ubiquitous resources (devices, sensors, interfaces, etc).

The *Service-Oriented Architecture* (SOA) [9][10] has attracted great attention firstly in the enterprise system integration. The concept of the SOA is well applied to ubiquitous/pervasive environment to achieve the programmatic interoperability among heterogeneous and distributed devices. Wrapping proprietary operations and protocols by Web services provides loose-coupling and platform-independent access methods for external software that uses the devices. The adaptive services can be implemented by statically or dynamically composing the existing services based on given requirements [4]. Several studies have been reported on the service orientation of ubiquitous applications (e.g., home networks [6][7] and sensor networks [8]).

As more and more ubiquitous devices become available as SOA services, the next challenge is how to manage such service resources efficiently and effectively. Our key idea is to borrow the idea of the *cloud computing* [11][12], regarding everything as a service (XaaS).

For the various ubiquitous resources deployed in the Web, different requirements will be expected from different clients. Expert programmers (or hackers) developing small custom applications may just want to use sensors and devices as they are. Developers and providers of large-scale adaptive applications may require more sophisticated environment or tools, which can dynamically discover necessary service resources based on contextual information. End users may want to use the adaptive applications as "tailor-made" services without concerning whatever resources are used. The above hierarchy of requirements fits well the structure of IaaS, PaaS and SaaS in the cloud computing [12].

Borrowing the concept of the cloud, this paper proposes an architecture of *ubiquitous cloud*, to manage service resources for adaptive ubiquitous applications and services. The key components of the proposed cloud are:

1) **Service Resource Registry:** Situated in the IaaS layer, it provides an infrastructure making ubiquitous resources available as atomic Web services.
2) **Adaptive Resource Finder:** Situated in the PaaS layer, it tries to find "optimal" service operations for users and environments, based on given requirements as well as the current user/environmental contexts.
3) **Context Manager:** Situated in the PaaS layer, it gathers various contextual information from real world, using the underlying sensor services.
4) **Service Concierge:** Situated in the SaaS layer, it recommends and executes published adaptive applications as tailor-made services for end users.

The development of the ubiquitous cloud is currently under way. In this paper, we especially discuss the details of the service resource registry and the adaptive resource finder. To show the effectiveness, we also conduct an experiment of developing a practical adaptive service with and without the ubiquitous cloud. As a result, it is shown that the development effort is significantly reduced.

II. Preliminaries

A. Service-Oriented Ubiquitous Network

In this paper, we assume a future ubiquitous network, where every object is deployed as an (atomic) SOA service

978-1-61284-936-2/11/$26.00 ©2011 IEEE

(e.g., [6][7][8]). Various types of ubiquitous devices are uniformly abstracted as platform-independent *service resources*, defined by standard interface description (e.g., WSDL) and access protocols (e.g., REST, SOAP). Typical service resources include house-hold appliances (TVs, DVDs, air-conditioners, lights), sensors, cars, user-interfaces (displays, microphones), telephones, PDAs, etc.

A service resource provides a set of *service operations* each of which implements a functional behavior of the resource. Integrating service operations from various resources achieves value-added *composite services*. For instance, orchestrating a TV, a DVD, a curtain, lights and speakers implements a DVD theater service, where a user can watch movies in a theater-like atmosphere at home. The composition of the service resources can be done statically in design time, or dynamically during run-time, depending on the target applications.

B. Adaptive Ubiquitous Services

Within the service-oriented ubiquitous network, we define an *adaptive ubiquitous service* as a composite service that can adapt its behaviors dynamically to user's requirements and varied contexts. A simple example is an "adaptive message delivery service". For a given message and a recipient address, the service chooses an appropriate transport of the message (e.g., chat, e-mail, phone, voice from speakers, TV subtitles, etc.), according to the status of the recipient. Another example is a "room environment relocation service", which saves an environment status of a room and restores the status in another room as a user moves.

One of the most challenging issues in developing adaptive services is how to find and use the service resources appropriate to the requirements and the contexts. This is a quite different point from the conventional services where all the service resources are determined statically at design time. Hence, we aim to support service developers as well as end users in this respect.

The goal of this paper is to provide a powerful mechanism of service resource management at different levels of abstractions.

III. Ubiquitous Cloud

To achieve the goal, we present the *ubiquitous cloud* to provide useful services that manage service resources in the service-oriented ubiquitous network.

A. Stakeholders

We suppose in the future that the service-oriented ubiquitous network will be used by various kinds of stakeholders. Therefore, the cloud must be designed so as to correspond to service resource managements at different levels of abstractions. For the moment, we assume the following clients:

- **Ubiquitous Resource Providers (URP):** People who want to provide own ubiquitous resources in the service-oriented ubiquitous network.
- **Custom Applications Developers (CAD):** People who want to develop their own applications by using available ubiquitous service resources as they are.
- **Adaptive Services Developers (ASD):** People who want to develop large-scale adaptive services that dynamically use service resources based on given requirements and contexts.
- **Adaptive Service Providers (ASP):** People who want to provide adaptive applications as commercial or non-commercial services for the end users.
- **End Users (EU):** People who want to use the published adaptive services as tailor-made services.

B. Architecture

To satisfy the stakeholders, the proposed ubiquitous cloud adopts a three-layered architecture, where each layer respectively corresponds to the IaaS, PaaS and SaaS of the conventional cloud. Figure 1 shows the architecture of the ubiquitous cloud.

- **Infrastructure Layer:** Provides fundamental features to make various ubiquitous objects available as service resources in the network. The infrastructure layer is supposed to be used by URP and CAD.
- **Platform Layer:** Provides development platforms to facilitate development of adaptive ubiquitous services. The platform layer is supposed to be used by CAD as well as ASD.
- **Application/Service Layer:** Provides showcases where adaptive services are deployed and provided. The Application/Service layer is supposed to be used by ASP and EU.

Within the three-layered architecture, we are currently developing the following four key components:

1) **Service Resource Registry:** Situated in the infrastructure layer, the service resource registry stores metadata explaining service resources that are registered by the URPs. It also supports the CADs (as well the ASDs) to identify service resources and operations by various attributes: device class, operation type, physical location, purpose, users, etc.
2) **Adaptive Resource Finder:** Situated in the platform layer, the adaptive resource finder recommends most appropriate service operations based on given requirements as well as the current contexts. It understands the requirements, and transforms the requirements into concrete queries for the service resource registry. The results of the queries are filtered by the contexts gathered by the context manager (explained below). Thus, the adaptive resource finder works as a powerful tool for the ASDs and the CADs to implement the adaptive services.
3) **Context Manager:** Situated in the platform platform layer, it gathers various contextual information from

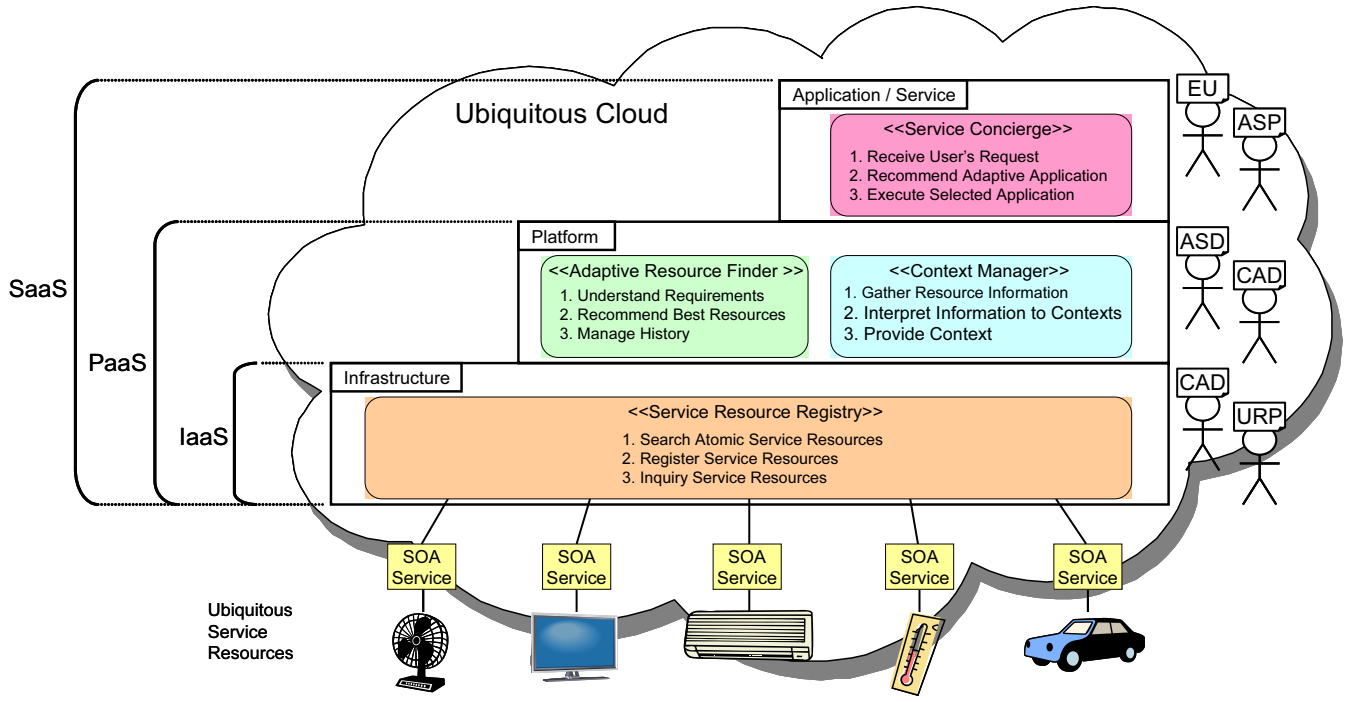

Figure 1. Ubiquitous Cloud for Adaptive Ubiquitous Services

real world, using the underlying sensor services. The gathered information is interpreted as explicit contexts characterizing user's status, requirements, tastes, environmental situation. The contexts are extensively used by the adaptive service applications as well as the adaptive resource finder. Thus, the context manager supports the ASDs and the CADs to obtain contextual information easily.

4) **Service Concierge:** Situated in the application/service layer, the service concierge provides a showcase for the developed applications and services, which are deployed by the ASP. It also recommends and executes published adaptive applications as tailor-made services of the EUs. Thus, the EU can use the services without concerning whatever service resources are used. Based on user's request, the behaviors of the services are dynamically determined.

We should note that the above four are just *key* components for the adaptive ubiquitous services, but not the complete set for the ubiquitous cloud. The cloud should evolve to contain more components, according to applications and purposes. The development of the four components is currently under way. Due to limited space, we especially focus on the details of the Service Resource Registry and the Adaptive Resource Finder in the following sub-sections.

C. Service Resource Registry

The Service Resource Registry stores meta-data of service resources in the service-oriented ubiquitous network. The registry is intended to be used for developers and applications to discover desired service resources and operations efficiently. As for a similar technology, UDDI (*Universal Description, Discovery and Integration*)) [15] is a well-known service registry for Web services.

However, we consider it difficult to use UDDI directly to implement our registry. The reason is that the UDDI mainly focuses on business and Web services in the IT world only, whereas our targets are physical ubiquitous objects in the real world interfacing the IT world.

In our context, an object is associated with a physical location and an explicit owner. An object could be an interface between the IT and real worlds. An operation of the object may work as *information source* from the real world to the IT world (e.g., sensors), or work as *information sink* (e.g., appliances, actuators) from the IT to the real.

Taking these issues into consideration, we have designed and implemented a service registry, called *UBI-REGI*, for ubiquitous service resources. The UBI-REGI is currently deployed in an actual home network system, CS27-HNS [7], developed by our research group. Figure 2 depicts an ER diagram of the UBI-REGI. A box represents an entity (i.e., table), consisting of an entity name, a primary key and attributes. We enumerate instances below each entity to support understanding. A line represents a relationship between entities, where $+\!\!-\!\!\in$ denotes a parent-children relationship, and $+\!\!-\!\!\cdots$ denotes a reference relationship.

In our UBI-REGI, a service resource (i.e., a ubiquitous object) is explained by a name, a device class, an end-point

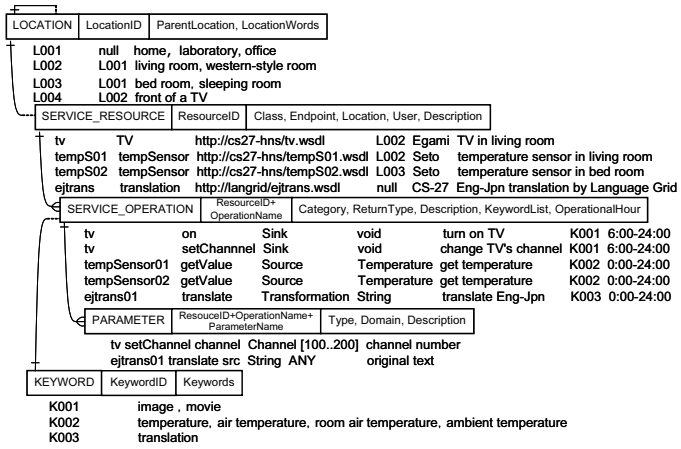

Figure 2. Data Model of Service Resource Registry

of Web service, a physical location, a user (owner) and a description. A physical location is characterized by location words, and is grouped by a parent location. A resource has a set of service operations (i.e., methods), each of which is characterized by a name, an operation category, a return value, a description, keywords representing its purpose, an operational hour. An operation takes parameters.

Wrapping the UBI-REGI as a back-end database, we implement the service resource registry as a collection of Web services (Web-APIs) for registering, searching and inquiring meta-data in the UBI-REGI. A typical API for the resource providers (URPs) is `registerServiceResource()`, while APIs for developers (CAD, ASP) include,

```
- findServiceResourcesByClass()
- findServiceResourcesByLocation()
- findServiceResourcesByUser()
- findServiceResourcesByOperation()
- getServiceResourceById()
- findServiceOperationsByCategory()
- findServiceOperationsByPurpose()
- findServiceOperationsByResource()
- getServiceOperationById()
```

The service resource registry has been implemented using the following technologies.

- Implementation: Java EE 6 SDK
- Database Server: MySQL 5.1.36
- Web Server: Apache Tomcat 5.5
- Web Service Engine: Apache Axis 2.1.3

D. Adaptive Resource Finder

The Adaptive Resource Finder aims to discover "optimal" service resources and operations for a given requirement from a client. The primary requirements for the adaptive resource finder is (R1) to understand the client requirement, and (R2) to recommend suitable service resources and operations for the client.

There would be many ways to define the data structure representing the client requirement. To capture the requirement from as many viewpoints as possible, we are currently using a *request template* to express a requirement from perspectives of WHAT, WHO, WHEN, WHERE, WHY, and HOW. The WHAT attribute should contain keywords to characterize what the main purpose is. The WHO contains information of the subjective user as well as the objective users of the recommended service. The WHEN is the desired date and time for the service, and the WHERE specifies the desired location. The WHY represents the background and motivation for the request. The HOW contains the concrete means on how the service should be realized (if any). Each attribute can be empty if there is no special request. An example of the request template is as follows:

```
REQ-ID: r0001
WHAT:   cool the room
WHO:    Koichi Egami for himself
WHEN:   now
WHERE:  living room
WHY:    -
HOW:    -
```

For a given request, the adaptive resource finder *translates* the request into concrete API calls of the service resource registry. For instance, `findService OperationByPurpose()` is used to find operations to achieve the purpose in the WHAT attribute. Also, to locate available resources within the place specified in the WHERE attribute, `findServiceResourcesByLocation()` is useful. Then, the adaptive resource finder executes the API calls and obtains information of the service resources and operations matching the request.

The next problem is to examine which resources and operations are more suitable for the *context*. We use the term context to represent any information characterizing the situation of users and surrounding environments. Typical contexts include user's status, mood, preferences, location, environments (temperature, humidity, brightness, etc.). Usually, such context is not explicitly specified in the request template. However, sorting discovered service operations by *suitability* to the context achieves good recommendation.

As an example, let us consider the above request template `r0001`. Suppose that two operations `window.open()` and `airConditioner.cooling()` are found from the service resource registry. The former operation should be recommended first if the season is winter or Koichi Egami likes energy-saving behaviors. If it is in summer and Koichi hates being very warm, the latter should be preferred.

Although we do not describe the details in this paper, we assume that the Context Manager (See Figure 1) takes all responsibilities of gathering and managing the context. After the adaptive resource finder obtains the available resources and operations from the service resource registry, it then asks the context manager to get the relevant contexts.

Based on the context, the finder puts the ranking for each operation, and recommends the operations to the client. To implement the recommendation, we consider to use the existing recommendation algorithms (e.g., [13][14]).

The adaptive resource finder can also execute a recommended service operation as a proxy of the client. As a supplementary feature, the adaptive service manages *history* of the recommendation, with which the application can cache the previous searching results for better performance. The history can also be used for investigating statistics of user's preferences. Similar to the service resource registry, we have designed and implemented the adaptive resource finder as a web service providing the following APIs.

```
- findAdaptiveOperations(Request req)
- executeOperation(ServiceOperation op)
- getAllRequestHistory()
- inqueryRequestHistory(RequestID req_id)
- clearRequestHistory(RequestID req_id)
```

The Web service has been implemented with the same technologies used for the service resource registry.

IV. Experimental Evaluation

We have conducted an experiment developing a practical adaptive service, "room environment relocation service". In the experiment, two versions of the service were developed WITH and WITHOUT the ubiquitous cloud, in order to see how the cloud facilitates the development.

A. Room Environment Relocation Service

The room environment relocation service saves an environment status of a room and tries to restore the status in another room. As a user enters a room, the service autonomously adapts the environment of the room to the previous one, using available ubiquitous resources.

In the experiment, the service has been implemented using our actual home network system, CS27-HNS [7]. As shown in Figure 3, there are two rooms: a living room and a Japanese-style room. The living room contains a ceiling light, a plasma TV, an air-conditioner and an air-cleaner, whereas the Japanese-style room is equipped with an analog TV, a table light, and an air-conditioner. All the appliances are connected to the network, and can be used as SOA service resources.

The service relocates a room environment from a living room to a Japanese-style room and vice versa. More specifically, as a user moves from a room (source room) to another (target room), the service saves the status of every appliance in the source room. Then, the service tries to restore the status using available appliances in the target room. Since the appliances in the two rooms are not necessarily the same, the relocation requires an adaptive resource discovery.

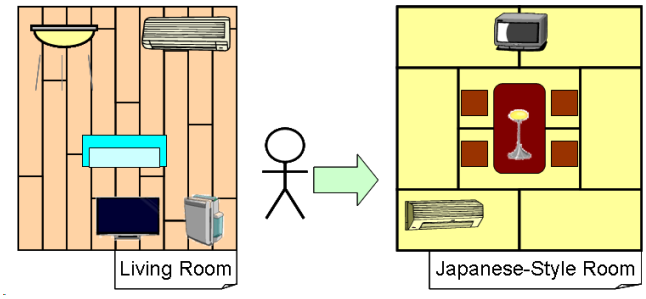

Figure 3. Room relocation service

B. Implementation of Service

In the experiment, we have implemented two versions of the room environment relocation service. One version was developed by the conventional technologies (labeled by Conventional-Ver.), another was developed using the proposed ubiquitous cloud (labeled by Cloud-Ver.). Both versions were written in Java, and the only one-way relocation from the living room to the Japanese-style room was considered (just to save the time of the experiment).

The relocation is basically performed as follows.

- Step1: Get a list l_1 of appliances in the source room.
- Step2: Save status of every appliance in l_1.
- Step3: Get a list l_2 of appliances in the target room.
- Step4: For every appliance in l_2, find any appliance operation that can restore the saved status. Put the operations in a list l_3.
- Step5: Execute all the operations in l_3 to restore the environment status.

In the Conventional-Ver., all above steps were implemented within a local application. The information of all service resources in the CS27-HNS was stored in a local MySQL database with a custom schema. Every procedure to find resources at Steps 1, 3, or 4 was implemented as a custom algorithm with SQL and OR mapper (iBatis). The total number of lines of code was 3327.

On the other hand, the Cloud-Ver. delegated most of the tasks to the ubiquitous cloud. Specifically, it used the service resource registry for Steps 1 and 3, and it used the adaptive resource finder for Step 4. Using the ubiquitous cloud, the Cloud-Ver. does not need to have the database, or sophisticated algorithms to locate the resources. The total number of lines of code was 632.

C. Discussion

Table I summarizes the comparison between the two versions of the implementation, from the viewpoints of the service resource management. By delegating complex tasks to the ubiquitous cloud, the size of Cloud-Ver. was significantly reduced compared to the Conventional-Ver. This implies that the ubiquitous cloud can achieve much less developing effort and much more reliability.

Table I
COMPARISON OF TWO VERSIONS OF ROOM ENVIRONMENT RELOCATION SERVICE

	Application Size (LOC)	Storing Resource Info.	Searching Resources	Adding, Deleting, Changing Resources
Conventional Ver.	3327	Local MySQL database	Custom algorithm with SQL and OR mapper	Update local database Rebuild application as new version
Ubiquitous Cloud Ver.	632	Delegate to cloud (service resource registry)	Delegate to cloud (adaptive resource finder)	Update service resource registry No change required in application

Moreover, using the ubiquitous cloud instead of proprietary resource management schemes improves the *portability* and *scalability* of the application. Even if new service resources are added, deleted or modified, the application can keep working without any change. Also, even when the number of resources becomes huge, it is unnecessary for individual applications to manage such a huge number of resources locally.

One major drawback is that a crash of the ubiquitous cloud leads to *annihilation* of all applications and services depending on the cloud. This is due to the centralized structure of the cloud computing. The reliability of the ubiquitous cloud itself must be guaranteed by other techniques.

V. CONCLUSION

In this paper, we presented a novel platform called *ubiquitous cloud* for adaptive ubiquitous applications and services. We also conducted an experiment of developing a practical adaptive service with and without adaptive resource finder. As a result, development efforts were significantly reduced by using the resource finder.

The development of the four components of the ubiquitous cloud is currently under way. In the future work, we will develop the remaining components, context manager and service concierge. Additionally, sice the current adaptive resource finder can find the resources by only class, location and user, we extends the resource finder to support searching by free keywords.

ACKNOWLEDGMENT

This research was partly supported by the Japan Ministry of Education, Science, Sports, and Culture, Grant-in-Aid for Young Scientists (B) (No.21700077) and Research Activity Start-up (No.22800042).

REFERENCES

[1] S. Aoki , M. Ito, J. Yura, J. Nakazawa, K. Takashio, and H. Tokuda, "u-Photo Mobile: Interacting with Smart Environments via Clickable Photos on Mobile Phones," International Conference on Intelligent Environment, Vol.2, pp.327–334, July 2009.

[2] K. Carey, D. Lewis, S. Higel, V. Wade, "Adaptive Composite Service Plans for Ubiquitous Computing," 2nd International Workshop on Managing Ubiquitous Communications and Services (MUCS 2004), December 2004.

[3] T. Cottenier, T. Elrad, "Adaptive Embedded Services for Pervasive Computing," Workshop on Building Software for Pervasive Computing - ACM SIGPLAN conf. on Object-Oriented Programming, Systems, Languages, and Applications, 2005.

[4] A. Urbieta, G. Barrutieta, J. Parra, A. Uribarren, "A Survey of Dynamic Service Composition Approaches for Ambient Systems,"

[5] R. Harbird, S. Hailes, C. Mascolo, "Adaptive Resource Discovery for Ubiquitous Computing," 2nd Workshop on Middleware for Pervasive and AdHoc Computing, 2004.

[6] J. Bourcier, A. Chazalet, M. Desertot, C. Escoffier, and C. Marin, "A dynamic-soa home control gateway," in *Proc. the 3rd IEEE International Conference on Services Computing (SCC)*, pp. 463–470, 2006.

[7] M. Nakamura, A. Tanaka, H. Igaki, H. Tamada, and K. Matsumoto, "Constructing home network systems and integrated services using legacy home appliances and web services," *International Journal of Web Services Research*, vol. 5, no. 1, pp. 82–98, 2008.

[8] J. King, R. Bose, H. i Yang, S. Pickles, and A. Helal, "Atlas: A service-oriented sensor platform hardware and middleware to enable programmable pervasive spaces," in *Proc. the 31st IEEE Conference on Local Computer Networks (LCN)*, pp. 630–638, 2006.

[9] M. P. Papazoglow, D. Georgakopoulos, "Service Oriented Computing," Communications of the ACM, 46(10), 25-28, 2003.

[10] T. Erl, "SOA Principles of Service Design," Prentice Hall, 2007.

[11] I. Foster, et al., "Cloud Computing and Grid Computing 360-Degree Compared," Grid Computing Environments Workshop (GCE 2008), p. 60-69, 2008.

[12] "What is Cloud Computing?," Whatis.com, http://searchcloudcomputing.techtarget.com/sDefinition/0,,sid201_gci1287881,00.html, 2008.

[13] A. Costa, R. Guizzardi, G. Guizzardi, and J. Filho, "COReS: Context-aware, Ontology-based Recommender System for Service Recommendation," in *Proc. International Conference on Advanced Information Systems Engineering (CAISE)*, 2007.

[14] Y. El Bouzekri El Idrissi and R. Ajhoun, "Context-based Services Selection and Recommendation Through P-learning Platform," in *Proc.International Conference on Information Technology based Higher Education and Training (ITHET)*, pp. 108–113, 2010.

[15] UDDI, http://uddi.xml.org/