International Journal of Multimedia and Ubiquitous Engineering
Vol. 4, No. 4, October, 2009

Ubiquitous Secretary: A Ubiquitous Computing Application Based on Web Services Architecture

Salmin Sultana[1], Rezwana Karim[2], Rifat Shahriyar[3], Md. Mostofa Akbar[4], and Sheikh Iqbal Ahamed [5]

Bangladesh University of Engineering & Technology[1, 2, 3, 4],
Marquette University[5]
salmin01010@yahoo.com[1], nawrin01022@yahoo.com[2],
rifat@cse.buet.ac.bd[3], mostofa@cse.buet.ac.bd[4],
sheikh.ahamed@marquette.edu[5]

Abstract

Due to the extensive availability of wireless internet connectivity and low cost light weight mobile devices, an omnipresent customizable service is not a vision anymore. With enhanced wireless connectivity, the handheld mobile devices (PDA, smart phone, cell phone, etc.) can act as a digital personal secretary, being employed in all aspects of life for the purposes ranging from accessing wide range of information to performing various types of activities with greater ease and comfort. If this type of service can be made a reality, it can be used by different types of users in different fields such as education, tourism, shopping or business, at any time and at any place. Here we present Ubiquitous Secretary, which is designed and developed to accomplish the above objectives. Ubiquitous computing applications developed so far are all independent and rich in their own domains but they suffer from the problem of non-inter operability and non scalability. Ubiquitous Secretary is developed upon the web services architecture that easily solves the interoperability and adaptability problem between different ubiquitous computing applications. We developed a prototype and presented how the implementation of the prototype satisfies the characteristics and features of Web Service based architecture.

Keywords: *Ubiquitous Secretary, Ubiquitous Computing, Web Services Architecture*

1. Introduction

Ubiquitous Computing, known as the third paradigm of computing introduces the age of calm technology [1] and the notion of one person, many computers as technologies weave themselves into the fabric of everyday life. It is the trend towards increasingly ubiquitous, connected computing devices in the environment, a trend being brought about by a convergence of advanced electronic - and particularly, wireless technologies and the Internet [2].

The handheld mobile devices such as PDA, smart phone, cell phone, etc with the help of enhanced wireless connectivity can act as a digital personal secretary. It can be employed in all aspects of life to perform various types of activities with greater ease and comfort by accessing wide range of information. With these devices the users can receive services like-exploring a city, searching for any specific information, assistance in meeting user demand, booking hotel rooms, receiving room services, finding route to a new place and many more. These benefits undoubtedly require creating a system that is pervasively and unobtrusively

embedded in the environment, completely connected, intuitive, effortlessly portable, and constantly available.

Conventional middleware systems like CORBA [3], DCOM [4], MOM [5] and RMI [6] [7] have been used to address the challenges imposed by ubiquitous computing environments. All these systems have given rise to a wide variety of architectures. Ubiquitous computing applications developed so far are all independent and rich in their own domains but the services and benefits provided by one application is not accessible by the others. They also suffer from the problem of non-inter operability and non scalability. Besides these, most of their underlying architectures are too complex and heavy for mobile devices. Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. Web services have been used in the past for B2B integration and inter or intra enterprise communication. Addressing all these issues, we explore the idea of a web service based simple and unique architecture of ubiquitous computing applications. We have extended the concept of Web Service to create another layer of abstraction over the different existing architectures and expose their functionalities as services. The architecture easily solves the interoperability and adaptability problem between different ubiquitous computing architectures and provides standard protocols for communication and service composition, description and discovery purpose. In this paper, we present architecture for ubiquitous computing applications based on the notion of Web service. We have also emphasized on the design development and deployment of Ubiquitous Secretary - the first prototype of a ubiquitous computing paradigm based on this architecture.

The outline of this paper is as follows: We provide the short descriptions of the related works in Section 2. Some scenarios and Pseudo solutions are described in Section 3 and 4. Proposed architecture of is described in Section 5 followed by the design features in Section 6. Development of Ubiquitous Secretary and its evaluation is presented in Section 7 and 8. Our future research direction and concluding remarks are in Section 9.

2. Related Works

All Several Ubiquitous Computing projects have been evolved in the past few years. Gaia [8] provides support for mobile user-centric active space applications. It manages the resources and services of an active space. It provides services for location, context, events and repositories with information about the active space. The current implementation of Gaia uses CORBA. However, it is possible to port Gaia to other communication middleware architectures including SOAP, RMI or customized implementations. Some of the Active space applications developed for Gaia prototype are: iCalendar (task scheduler used to schedule seminars and meetings), attendance (Records participants for a task), mPPT (displays multiple synchronized PowerPoint presentations), TickerTape (scrolls information around the room displays) etc.

Project Aura [9] gives a solution to user mobility problem based on the concept of personal Aura. The intuition behind a personal Aura is that it acts as a proxy for the mobile user it represents. When a user enters a new environment, his or her Aura marshals the appropriate resources to support the user's task. Task Manager of Aura, called Prism, embodies the concept of personal Aura. The Context Observer provides information on the physical context and reports relevant events in the physical context back to Prism and the Environment Manager. Environment Manager embodies the gateway to the environment and Suppliers provide the abstract services that tasks are

composed of (text editing, video playing, etc). Furthermore, an Aura captures constraints that the physical context around the user imposes on tasks.

The One.World [10] architecture employs a classic user kernel split: Foundation and system services run in the kernel and Libraries, system utilities, and applications run in user space. The four foundation services directly address the three requirements of change, ad hoc composition, and pervasive sharing. They also provide the basis for the system service of our architecture, which in turn serve as common building blocks for pervasive applications. For example, the query engine provides the ability to search tuples by instantiating filters, structured I/O lets applications access stored tuples in environments; it supports the writing, reading, querying, and deleting of tuples. Migration provides the ability to move or copy an environment and its contents, including stored tuples, application components, and nested environments, either locally or to another device.

In the Oxygen project [11], 'intelligent space' occupied by cameras, microphones, displays, sound output systems, radar systems, wireless networks and controls for physical entities, were introduced. People can interact by using speech, hand gestures, drawing, and body movement.

In Conference Assistant [12], a prototype for assisting conference attendees in choosing presentations to attend, taking notes, and retrieving those notes was presented. They also discussed the important relationship between context-awareness and wearable computing.

All these projects suffer from some drawbacks. Gaia is too heavy for mobile devices as it is based on CORBA. The task manager of Aura is built upon RPC and CORBA and as it is an ongoing project some of the details are still very vague. One.World is criticized as it is built upon JVM and MS CLR. Also it uses tuples to define data models where XML could have been an excellent alternative.

3. Ubiquitous Scenarios

To better understand the need of Ubiquitous Secretary, it is helpful to describe some scenarios using available devices and services.

Scenario 1

Diana flies from Indonesia to Shanghai to attend the World Engineers Convention. When she arrives at Pudong International Airport, she finds with utter disappointment that the volunteers have forgotten to receive her. Nothing to worry about! The Ubiquitous Secretary in Diana’s PDA gives her present location as well as the location and route to one the three star hotels Jian Tiang just after she queries. In the hotel lobby, Ubiquitous Secretary automatically joins the hotel reception network and verifies Diana’s identification from her PDA through the Hotel Reception Service. Ubiquitous Secretary then provides a list of suitable rooms according to Diana’s preference. After Diana chooses her favorite room, the reception service transfers an electronic certificate into her PDA which authorizes the PDA to join the room network, open the door lock and control devices in the room etc. When Diana is nearing her room, her PDA joins the room network. After a long sleep, Diana gets up and decides to go out for a visit of Shanghai. Ubiquitous Secretary helps her to browse the list of famous places

of Shanghai and gets reliable and necessary information about them. She selects ocean Aquarium and Super Brand Mall for her visit.

Scenario 2

David is at home working on the organization of a conference in a remote place. He is gathering information on possible venues and getting budgets for catering. The web pages of some of the hotels include short videos featuring virtual visits to the premises and David already downloaded some of these. David is also taking notes on the spreadsheet concerning his appraisal of each venue along with the alternative catering budgets. David leaves home and heads to his office. Since David intends to continue working, his Ubiquitous Secretary, being capable of invoking all the services provided by Aura [9] sets up that task at David's office so that he can resume his work as soon as he is recognized entering the office: a web browser over the recently visited pages, the downloaded videos paused at the same places, and a spreadsheet containing all the entered figures. Since there is a big screen on the wall of David's office, that is preferred for video and web browsing, releasing Monitor for the spreadsheet.

Scenario 3

Professor John is supposed to give a talk on Ubiquitous computing. When he enters the classroom his Ubiquitous Secretary joins the room network and automatically uploads the slides to be presented in the PC of that room. The A/V devices in the room are automatically discovered and connected to Ubiquitous Secretary. During the talk, the audio and video feeds from the room are captured and saved. The audio and video can then be made available to the presenter and the students. Ubiquitous Secretary is a complete service provider for all of the above scenarios. Any service developer can develop customizable applications according to the user's need and make them accessible to the Ubiquitous Secretary.

4. Pseudo Solution

All the existing projects address the challenges of ubiquitous computing more or less but they are based on different architectural models. So, a service that is provided by Gaia cannot be accessed by AURA or One.World and vice versa. As all of these projects are huge and established, it is less likely that they will be re-factored to become interoperable. But before proposing the new architecture, let us examine whether it is possible to make them interoperable by simple modification. One way to solve the problem is to introduce a centralized middleware controlled by a third party [13]. All the existing architectures will have to agree on using and co-operatively maintaining a certain middleware platform.

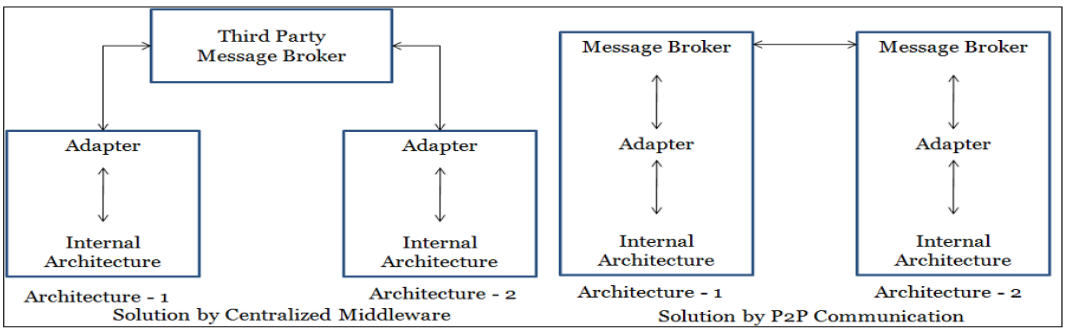

Figure 1. Existing Solutions

This may be implemented by deploying a specific message broker, a specific workflow system and a specific name and directory server. Ubiquitous computing applications intended for secure transactions between two parties may not want to use it due to lack of trust and the confidentiality of the transactions play against the idea of having a centralized middleware hosted by a third party. Another solution to the interoperability problem can be to address the problem in a point to point communication, by separately tackling the integration problem with each of the partners [13]. This means that two parties willing to communicate can agree on using certain middleware protocols and infrastructures. For example, they can both deploy a message broker and use it to send message to each other. But as a ubiquitous computing application of certain architecture will have to interact with many different applications of different architectures and each of them may require the use of different middleware platform, this leads to a scenario when a particular ubiquitous computing architecture will have to support many heterogeneous middleware. So the problem with the point to point communication is that it requires employing different brokers for different architectures which is not scalable and feasible.

5. Proposed Architecture

The architecture proposed by us tends to solve the interoperability problem by creating another level of abstraction over the current architectures. The architecture is based on Web Service. Web services work on the assumption that the functionality made available by the system will be exposed as a service. The proposed architecture is presented in the following Figure.

New ubiquitous computing applications can be built on this architecture. Also, it can be implemented over any existing ubiquitous computing architecture. It is mainly composed of two major components based on two aspects. The first aspect is related to the fact that web services are a way to expose internal operations so that they can be invoked through the web. Such an implementation requires the system to be able to receive requests through the

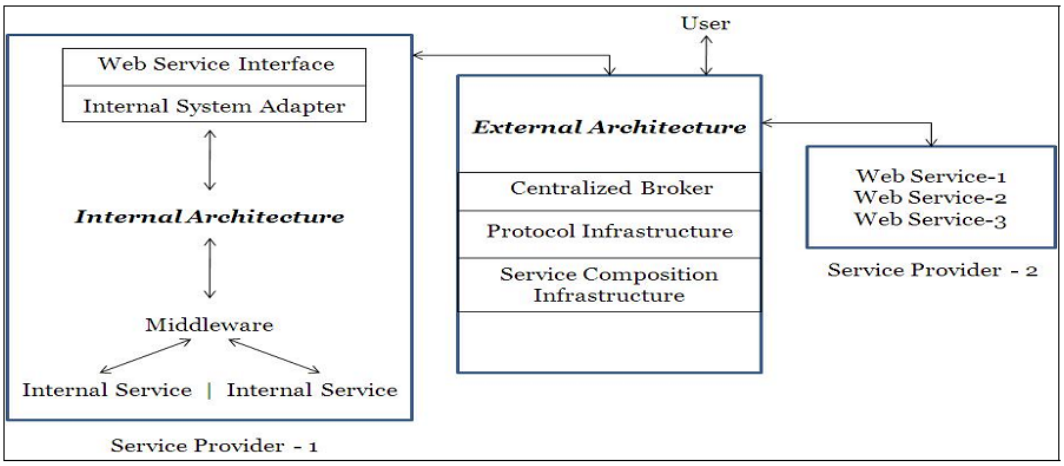

Figure 2. Proposed Web Service based Architecture

web and to pass them to the underlying system. In doing this, the problems are analogous to those encountered in conventional middleware. We will refer to such an infrastructure as internal middleware. Correspondingly, we will use the term internal architecture to refer to

the organization and structure of the internal middleware. The other facet of the architecture is represented by the middleware infrastructure whose purpose is to integrate different web services. We will refer to such an infrastructure as external middleware. Correspondingly, we will use the term external architecture to refer to the organization and structure of the external middleware.

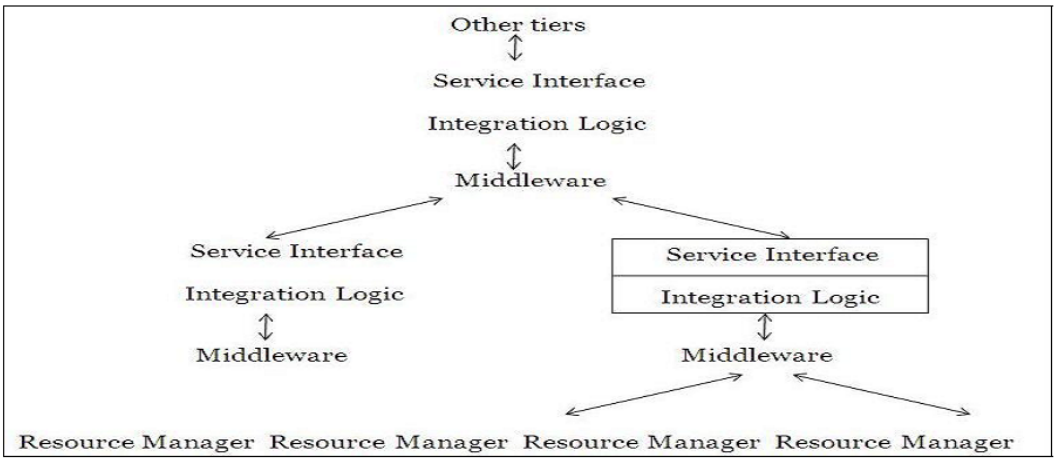

Figure 3. Conventional Middleware as an integration platform

Internal Architecture

The easiest way to understand the internal architecture is to view it as yet another tier on top of the other tiers of the enterprise architecture. Conventional middleware is used to build multi-tier architectures. In these architectures, individual programs or applications are hidden behind service abstractions that are combined into higher order programs or applications by using the functionality provided by the underlying middleware. The resulting higher order programs can in turn be hidden behind new service abstractions and can be used as building blocks for new services. Since the composition of service abstractions can be repeated spontaneously, the result is a multitier system in which services are implemented atop other services and basic programs. When multiple middleware instances are stacked on top of each other, the middleware used at each level does not need to be the same. The important point is to have compatible service abstractions or to make them compatible using wrappers. The middleware simply acts as the glue necessary to make all the components in a given level interact with each other to form services that can be used by clients or higher levels in the hierarchy. Although it is not strictly necessary, usually the basic components of each middleware instance reside on a Local Area Network (LAN) and the resulting application also runs on the same LAN. Web services or, better, the technologies supporting web services, play the same role as conventional middleware, but on a different scale. The basis for composition is service abstractions very similar in nature to those used in conventional middleware, so that implementing a web service essentially requires an extra tier on top of the others to enable access using standard web services protocols. The following Figure shows a typical example of such an internal architecture.

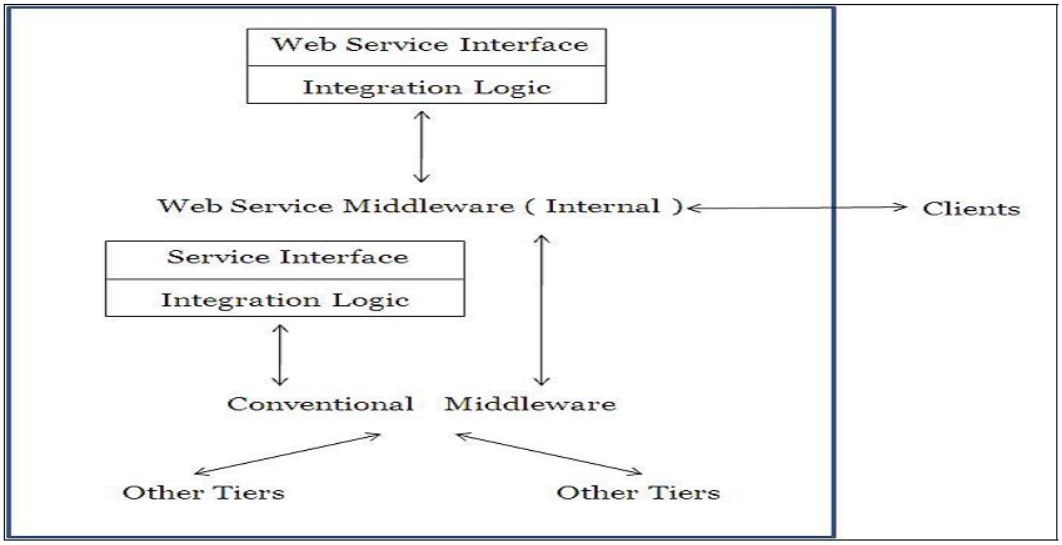

Figure 4. Internal Architecture

External Architecture

The external architecture has three main components:

1. *Centralized brokers*: These are analogous to the centralized components in conventional middleware that route messages and provide properties to the interactions (such as logging, transactional guarantees, name and directory services, and reliability). However, as we will see, in practice the name and directory server is often the only centralized component present in Web service based architecture.

2. *Protocol infrastructure*: This refers to the set of components that coordinate the interactions among Web services and, in particular, implement the P2P protocols whose aim is to provide middleware properties in those B2B settings where a centralized middleware platform cannot be put in place due to trust and privacy issues.

3. *Service composition infrastructure*: This refers to the set of tools that support the definition and execution of composite services.

What we have discussed until now is related to wrapping internal functionality as a Web service, and not to integrating these 'wrappers'. This aspect, which was addressed by message brokers and workflow management systems in conventional middleware, should be the job of the external middleware. However, it is not clear where this middleware should reside. Let us consider as an example the implementation of name and directory services. In LAN-based systems, the middleware and the applications developed using the middleware run next to each other. Thus, it is easy for the middleware to provide the necessary brokerage for name and directory services to all parties involved. In Web services, the parties can reside in different locations, and there is therefore no obvious place where to locate the middleware.

There are two solutions to this problem. One is to implement the middleware as a P2P system where all participants cooperate to provide name and directory services. Conceptually, this is a very appealing approach; but it is not obvious how to provide the degree of reliability and trustworthiness required in industrial strength systems. The other solution is to introduce

intermediaries or brokers acting as the necessary middleware. Assuming we find a site somewhere in the network that we can trust and that is reliable enough, the site could act as a name and directory server for Web services. Hence, we will use the second approach where the external architecture works as the coordinator among different Web services. Service publishing, service discovery and communication are the main tasks of this part. Service providers create Web services and define an interface for invoking them. They also generate service descriptions for those services. The service provider then makes its services known to the world by publishing the corresponding service descriptions in a service registry. When a service requester tries to find a service, it queries the service registry. The service registry answers with a service description that indicates where to locate the service and how to invoke it.

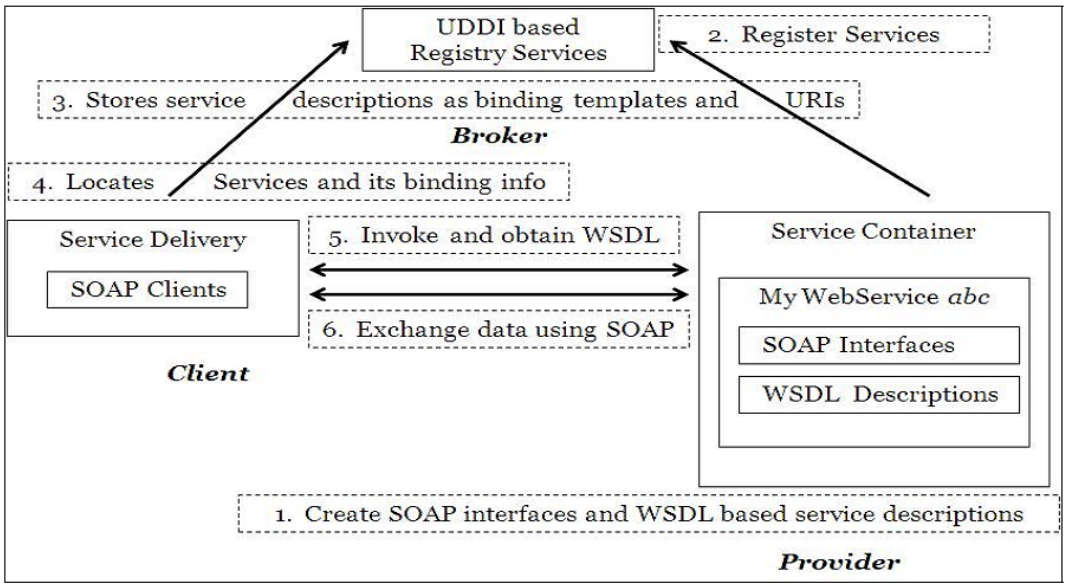

Figure 5. External Architecture

The sequence of action in our proposed architecture is enumerated below in terms of protocols and standards:

1. The service provider creates the Web service typically as SOAP-based service [14] interfaces for exposed applications. The provider then deploys them in a service container or using a SOAP runtime environment, and then makes them available for invocation over a network. The service provider also describes the Web service as a WSDL-based service description, which defines the clients and the service container with a consistent way of identifying the service location, operations, and its communication model.

2. The service provider then registers the WSDL-based service [15] description with a service broker, which is typically a UDDI registry [16].

3. The UDDI registry then stores the service description as binding templates and URLs to WSDLs located in the service provider environment.

4. The service requestor then locates the required services by querying the UDDI registry. The service requestor obtains the binding information and the URLs to identify the service provider.

5. Using the binding information, the service requestor then invokes the service provider and then retrieves the WSDL Service description for those registered services. Then, the service requestor creates a client proxy application and establishes communication with the service provider using SOAP.

6. Finally, the service requestor communicates with the service provider and exchanges data or messages by invoking the available services in the service container.

6. Design Features

The proposed architecture has the following features to note:

1. *Interoperability*: The proposed architecture does not preclude any programming model and is comprised of loosely-coupled components and their interrelationships.

2. *Simplicity*: It provides a simple mechanism for applications to become services that are accessible by anyone, anywhere, and from any device.

3. *Dynamic Discovery*: It enables dynamic location and invocation of services through service brokers.

4. *Integration with the World Wide Web*: The model is consistent with the current and future evolution of the World Wide Web and the architectural principles and design goals of the existing Web.

5. *Security*: The architecture provides a secure environment for online processes. It enables privacy protection for the users across multiple domains and services. Authentication and authorization mechanisms can be easily incorporated with this architecture using SSL enabling encryption of the message. Adopting open security standards like SAML [17], XML Encryption, XML Signature, or XACML [18] may be a solution.

6. *Scalability and Extensibility*: The architecture is sufficiently extensible to allow for future evolution of technology and new Ubiquitous Computing architectures.

7. *Lightweight*: As the architecture assumes communication via XML the client devices only should have the ability to send or receive XML messages; especially suitable for mobile devices with limited memory and processing capability.

7. Development of Ubiquitous Secretary

Underlying Architecture

Ubiquitous Secretary is developed on the Web Service based architecture [19] shown in Figure. The architecture is composed of three main components: Service Provider, Service Broker and Service Requestor or User. Service providers create Web services and define an interface for invoking them. They also generate service descriptions for their services and publish them in a service broker i.e. registry server. A service may be an application built on this web services based architecture or on any other ubiquitous computing architecture. These individual services or applications are hidden behind service abstractions that are combined into higher order programs or applications by

using the functionality provided by the underlying middleware. When a service requester tries to find a service, it queries the service registry. The service registry returns a service description indicating where to locate the service and how to invoke it. Service requestor depends on two underlying components - Context Service and User Profile Management.

Context Service receives information regarding context of the user from GPS, other sensors and supplies those to service requestor. User Profile Management maintains the user profile containing user information, preferences task schedule etc. Service requestor uses this information during service invocation to get the desired result.

General Approach of Implementation

Whenever a person wants to use any service she needs to know the services provided by her current location. The information about all the services provided in that particular location are stored in a local registry server. A unique and publicly known universal registry server contains the inquiry URL and publishes URL of all the country wide registry servers. The Client module takes the current location information from GPS that works in background and finds out the corresponding country code. A web service RegURLService using this code fetches the inquiry and publish URL of the local registry server from the "universal registry server". The necessary information related to all the available services in the current location is fetched from the local registry server and shown to the user. When a service is invoked, its corresponding Web Service Description Language (WSDL) [15] file is fetched and parsed. Parsed service specific information such as serviceName, wsdlImplURI, endpointURL, inputPramaters etc. are then stored. An InputGUI window is dynamically created based on required input parameters, takes necessary input from users and requests the service. The service provider receives the input, generates output and sends back the results to the service requestor in vector form. An output window is then generated that shows the output in suitable format.

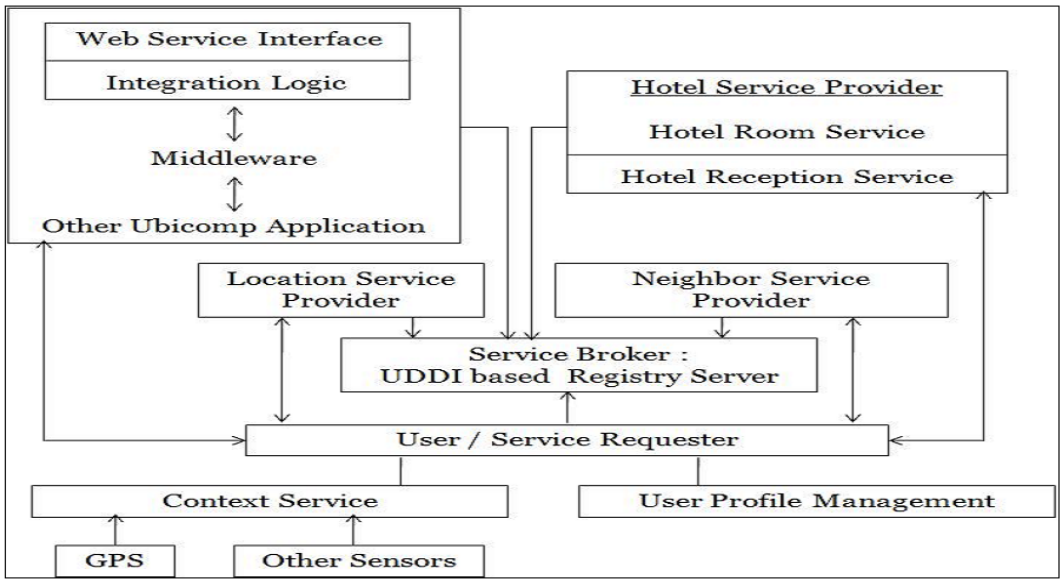

Figure 6. Web Service based Architecture

How Ubiquitous Secretary Satisfies All the Characteristics

Web Services itself supports Interoperability. As Ubiquitous Secretary is implemented based on web services, a service can be developed in any platform and any language but still can be invoked from the same client module. Different services can be developed by different developers but still can be integrated in the same registry server. The databases needed for different service are also decoupled and database servers can also be different; these are just dependent on the type of service and the developers' choice. From the developers' point of view, a service can be easily developed; they should just deploy the service in their own server to generate the WSDL file and publish the service in appropriate registry server. From the user's point of view, a service can be explored and invoked from anywhere and any device. Thus the paradigm ensures simplicity, scalability, flexibility and extensibility. Only the Client module resides and executes in user device and this module has tiny memory footprint. Our application is truly context sensitive. The services the user can invoke are entirely dependent on the user's current context and the service list is updated dynamically from place to place without any user intervention thus paving the way for dynamic discovery of services and context sensitivity. Though the services are distributed, they can be invoked from anywhere any device and the results of their invocation are available to the user within few seconds thus ensuring time efficiency. The information provided by the services are valid and secure as each service has to be published in the registry server, where the authority is responsible for validating and authenticating the service provider. Thus Ubiquitous Secretary ensures reliability here.

8. Evaluation

We evaluate the Ubiquitous Secretary in the following way:

1. Prototype Implementation
2. Cognitive walkthrough strategy
3. Performance measurement

Prototype Implementation

We developed a prototype of the Ubiquitous secretary. We use .NET Compact Framework 2.0 [20] and developed the application for Windows mobile. We use an ASP.NET Web site for creating XML Web services and MySQL as the database server. We use Microsoft provided Enterprise UDDI Services [21], a dynamic and flexible infrastructure for XML Web services. This standards-based solution enables to run ones own UDDI (Universal Description, Discovery, and Integration) directory for intranet or extranet use, making it easier to discover web services and other programmatic resources. Microsoft also offers UDDI client support through several tools including Visual Studio .NET. We used that to provide access to UDDI registry from within the application. To consume a Web service in a NET Compact Framework project, we need to add a Web Reference to the project. A Web reference enables a project to consume a Web service. When the Web Reference is added to your project, Visual Studio .NET automatically generates a proxy class with methods that serve as proxies for each exposed method of the Web service.

The core services provided by our Ubiquitous Secretary are:

1. *LocationService*: The LocationService informs the user of the current location.

2. *NeighborService*: NeighborService provides the user with the list of nearby facilities such as hotels, restaurants, museums, companies, parks and markets.

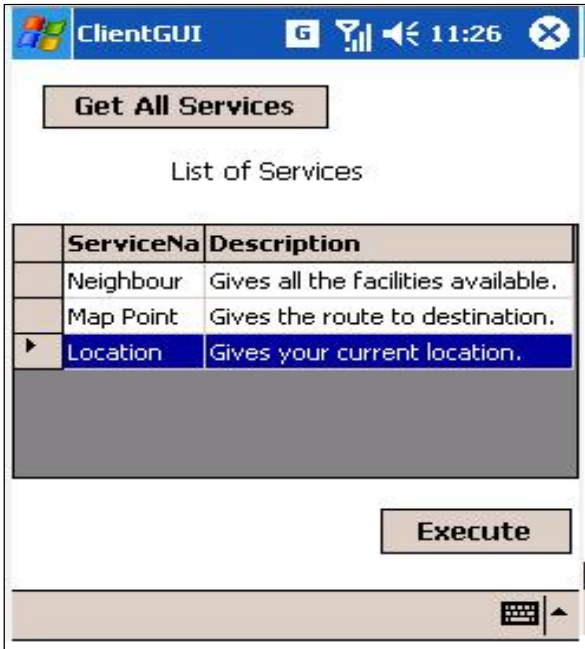

Figure : List of Services

Figure : Location Service

Figure : Map Point Service

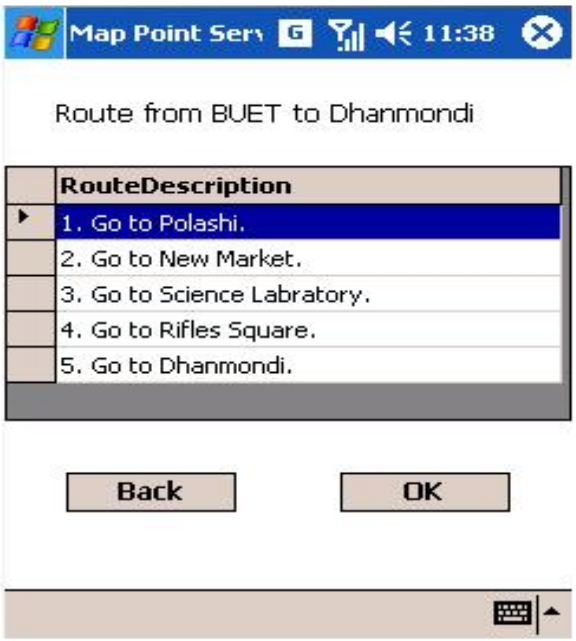

Figure : Route to Destination

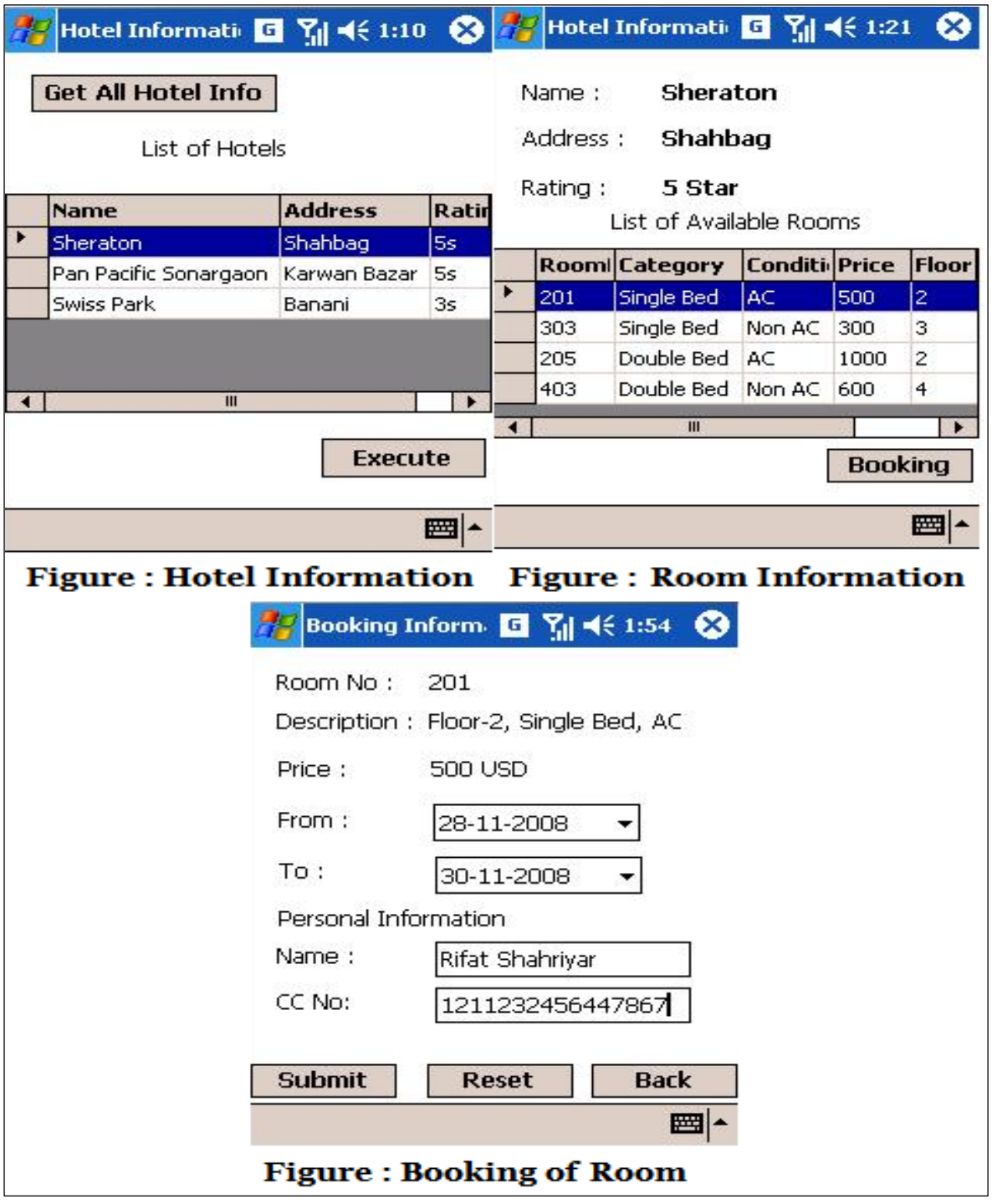

Figure 7. Screenshots of Prototype

3. *MapPointService*: It gives a route or location of a destination. After providing current location information and destination information, MapPointService successfully gives the user the location of the destination and how to get there.

4. *HotelReceptionService*: It provides the user with the available room list with their facilities.

5. *HotelRoomService*: The Hotel Room Service provides a user with the details of her room.

Some screen shots of the prototype are shown above.

Cognitive walkthrough strategy

Cognitive Walkthrough Strategy [22] encompasses one or a group of evaluators who inspect a user interface by going through a set of tasks and assess its understandability and ease of learning. To evaluate our Ubiquitous Secretary, we followed this strategy.

1. Who will be the users of the system? 2 Ph.D. students (Computer Science and Eng.), 2 graduate students (Computer Science and Eng.), 1 undergrad student (Chemical Eng.), 2 from general mass, 1 professor, 1 lecturer and 1 service holder were chosen as the users. We have tried to cover all type of end users.

2. What tasks will be analyzed? The services provided by our Ubiquitous Secretary were executed by the users. We have tried to select the tasks to be analyzed in such a way that no major task has been overlooked.

3. What is the correct action sequence for each task? First, we briefly explained the task sequences and process to get result. A questionnaire was given to the users. [23] The following Figures show the user's and application developer's satisfaction rating.

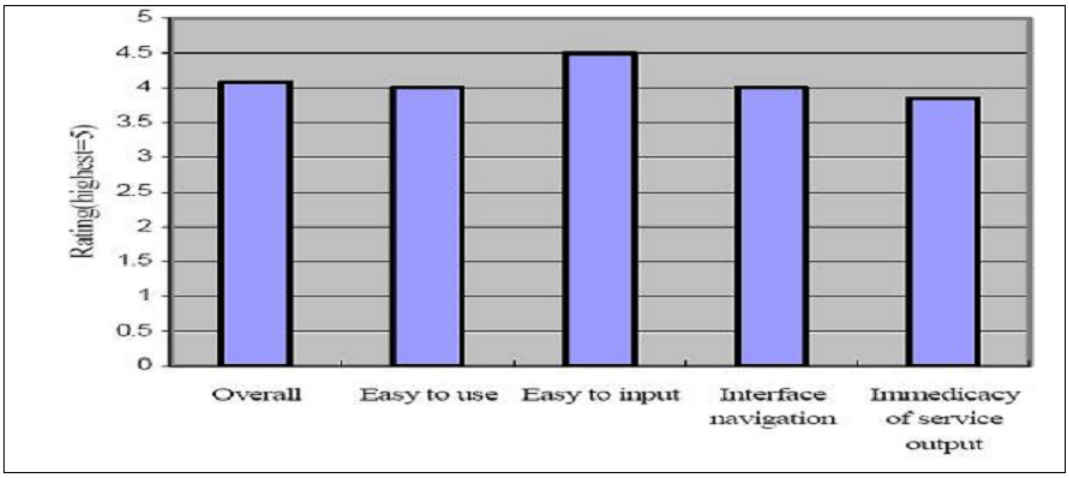

Figure 8. Rating by Users

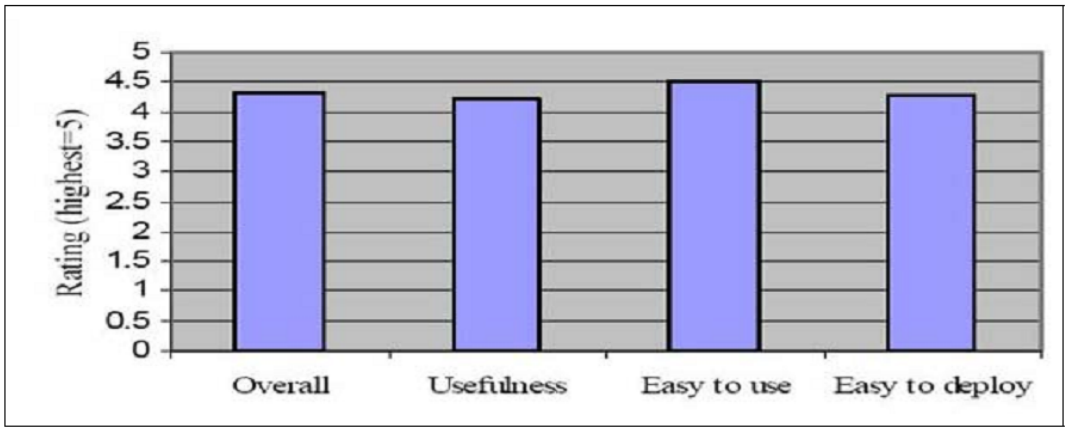

Figure 9. Rating by Application Developer

Performance Measurement

We have measured the time required for service discovery and different service invocation using US. The following Figure shows timing diagram for dynamic service discovery, Location service invocation and time to appear input form for MapPoint service invocation.

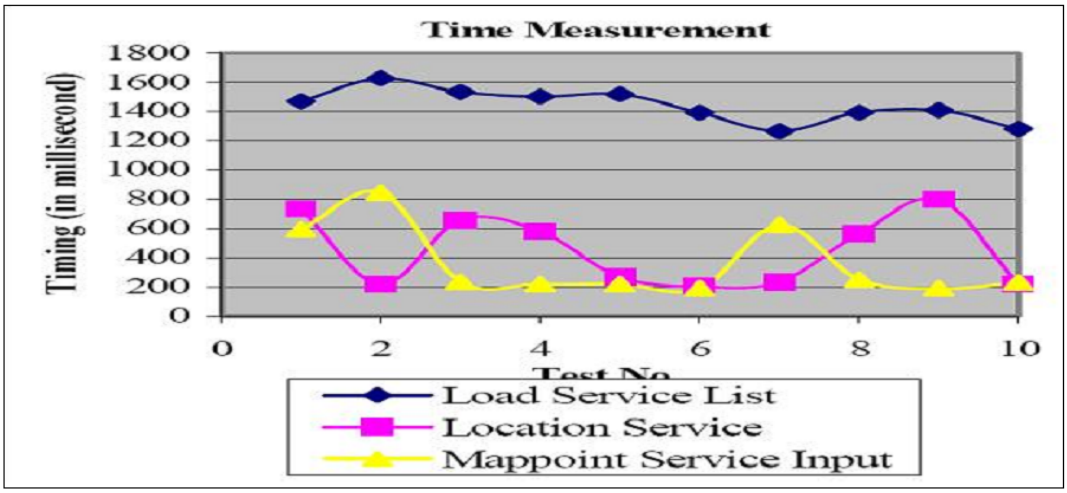

Figure 10. Time measurement of different Service invocation

The average time needed to discover services is approximately 1.44 sec, to invoke Location service is 0.45 sec and to appear input form for MapPoint service is 0.35 sec. The time needed for Neighbor Service, MapPoint service and HotelRoom service invocation given inputs are like Location service. Thus, Ubiquitous Secretary needs reasonable time to provide output which ensures its acceptability and user satisfaction.

9. Future Works and Conclusion

In this paper, we have shown that Ubiquitous Secretary - a ubiquitous paradigm based on Web services based architecture can be developed successfully while ensuring almost all the features as have been demanded by the proposer of the architecture [24] and all the characteristics of Ubiquitous Secretary as required in different fields. In Ubiquitous Secretary, the service publishing and discovery issues are easily handled by the existing standards like WSDL and UDDI. In terms of interoperability, scalability, robustness and security this paradigm developed on Web services based architecture seems to be a perfect example of ubiquitous computing applications. Although we have claimed the system to be secure, we have not introduced the concept of XML encryption, XML Signatures and SAML in our architectural design. The proposed architecture relies on the assumption that agents coming from trusted entities would not hoard system resources to pose denial of service threats. The Semantic Web efforts, especially with respect to the recent trend toward Semantic Web Services, aim at fully automating all the stages of the Web Services lifecycle. In the future we will incorporate User authentication by using signature or fingerprint and Semantic Web Services [25] to discover suitable services intelligently.

References

[1] M. Weiser and J. Seely-Brown, "The Coming Age of Calm Technology", eds. Copernicus, Heidelberg, Germany, 1998.

[2] Ubiquitous computing, http://en.wikipedia.org/wiki/Ubiquitous computing.

[3] Introduction to CORBA, http://java.sun.com/developer/onlineTraining/corba/corba.html

[4] DCOM Technical Overview, http://msdn.microsoft.com/en-us/library/ms809340.aspx.

[5] Message Oriented Middleware, http://www.tml.tkk.fi/Opinnot/Tik-110.551/1997/mqs.htm.

[6] Remote Method Invocation Home, http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp.

[7] Distributed Java Programming with RMI and CORBA,
http://java.sun.com/developer/technicalArticles/RMI/rmi _corba/

[8] M. Roman, C. Hess, R. Cerqueria, A. Ranganat, R.H. Campbell, and K. Nahrstedt, "Gaia: A Middleware Infrastructure to Enable Active Spaces", Mobile Computing and Communications Review, vol. 6, no. 4, pp. 65-67, June 2002.

[9] J. P. Sousa and D. Garlan, "Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments", Kluwer Academic Publishers, pp. 29-43, 2002.

[10] R. Grimm, "One.world: Experiences with a Pervasive Computing Architecture", IEEE Pervasive Computing, pp. 22-30, July-September 2004.

[11] MIT Project Oxygen, Pervasive Human-Centered Computing, "Project Overview", July 2005.

[12] Dey, A. K., Salber, D., Abowd, G. D., and Futakawa, M., "The Conference Assistant: Combining Context-Awareness with Wearable Computing", ISWC 1999.

[13] G. Alonso, F. Casati, H. Kuno and V. Machiraju, "Web Services: Concepts, Architectures and Applications", Springer-Verlag, 2004, pp. 123-148.

[14] Simple Object Access Protocol (SOAP) 1.1, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[15] Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl

[16] UDDI Version 3.0.2, http://www.uddi.org/pubs/uddi v3.htm

[17] Security Assertion Markup Language (SAML), http://en.wikipedia.org/wiki/SAML

[18] eXtensible Access Control Markup Language (XACML), http://en.wikipedia.org/wiki/XACML

[19] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, "Web Services Architecture", W3C Working Group Note, February 11, 2004.

[20] Microsoft .NET Compact Framework 2.0 http://blogs.msdn.com/netcfteam/

[21] UDDI Services, http://uddi.microsoft.com

[22] John Rieman, Marita Franzke, and David Redmiles,"Usability Evaluation with the Cognitive Walkthrough", CHI '95 Proceedings@ACM

[23] Usability Inspection: Cognitive Walkthrough, http://www.pages.drexel.edu/ zwz22/CognWalk.htm

[24] Dr. Md. Mostofa Akbar, Imranul Hoque, Sonia Jahid, "A Web Service Based Architecture for Ubiquitous Computing Applications", International Conference on Computer and Information Technology (ICCIT)-2005, pp. 710 - 715, Dhaka, Bangladesh, 2005 (www.iccit.org)

[25] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng.,"SemanticWeb Services", IEEE Intelligent Systems, 16(2):4653, March/April 2001.