An Evolvable Software Architecture for Managing Ubiquitous Systems and Devices

Syed Shariyar Murtaza, Choong Seon Hong

Dept. of Computer Engineering, Networking Lab.
Kyung Hee University, Korea

shariyar@networking.khu.ac.kr, cshong@khu.ac.kr

Abstract

This paper presents an evolvable software architecture for ubiquitous systems. It follows a modular and service oriented approach. By modular we mean, software is componentized and the components are distributed among the devices, available in an environment with respect to their computational strength. We argue that the components of the middleware are the specialized research areas with different subject orientation. Therefore, following this modular approach, would let the components of the software evolve in an object oriented fashion, while working together via service oriented approach. It would allow easy and independent configuration management, monitoring and control of the devices in the system.

Keywords: Web, URI, Semantic Web, Ubiquitous systems, Distributed and Scalable software architecture
Introduction

- Ubiquitous computing
  * A concept of fabricating computers in environments seamlessly.

- Software infrastructure for ubiquitous computing
  * Get information from the environment and control them.
  * Current infrastructure usually rely on proprietary protocols
  * Component reusability is rare.
  * Don't allow incremental development in the system.

- We propose a modular distributed software architecture
  * It distributes the software components into devices.
  * These components could be developed and updated independently without effecting the system.

1. Introduction

Ubiquitous computing can be thought of as opposite of virtual reality. It is the concept of fabricating computers into our regular working and living environments seamlessly. To make this vision a reality, hardware as well as software infrastructure is needed.

The main aim of a software infrastructure for ubiquitous computing is to get information from the environment with heterogeneous devices and provide means to control them.

Different types of software infrastructures for the ubiquitous systems have been developed in the in the past years, among which much of them are only concerned with one or more aspects in an ad hoc manner e.g. Context Toolkit[11] separates the low level sensing from high-level applications, and introduces a middleware layer, with some reusable components. The Solar system[12] architecture proposed a graph-based abstraction for context aggregation and dissemination. Gaia [13] a distributed middleware infrastructure that provides support for context aware agents in smart spaces. It uses CORBA,DAML and different logic reasoning and machine learning approaches. [7] integrates the context aware architecture with OSGi gateway. Pervasive Servers [9] integrates web server in every device and classifies them into two categories based on their functionality.

There have been several proposals for software infrastructures for ubiquitous computing, but such an infrastructure need to provide gradual improvement, allow the individual components to be able to improve independently. Gradual and independent improvement in a ubiquitous environment is necessary because the interfaces between components are changed frequently and it is difficult to assume the priori knowledge about the environment e.g. to make a product more attractive, a vendor would like to change the interfaces to their appliances.

In addition, most of them rely on a proprietary protocol, thereby set a barrier to interoperability of different systems, and exclude developers from reusing existing components. The agents or other applications can’t interoperate easily with other applications or agents based on other middleware. Also; if the agent is a mobile agent then for it to move around different devices requires the same platform. We overcome this by using Service oriented approach.

Another limitation of traditional middlewares is a finite set of different ontologies defined for that particular middleware or for some specific domain.
The URI based approach, in our system, promotes the use of ontology by individual users. With this approach, an infinite set of ontologies could be included. It means we can have different types of ontologies from variety of sources (the details have been discussed in the later sections). When a group of users share such a physical environment, new forms of sharing, cooperation and collaboration are possible, particularly by using shared ontologies.

We also need a standard accessing mechanism to access the appliances, so that we can make them collaborate. If each appliance supports a different means of access, ubiquitous computing environments will become too complex, and it will be very hard to manage robustly.

We have already proposed a mechanism for uniform identification and accessibility of objects [10], shown in above figure. Based on that we also propose a modular distributed middleware architecture, which distributes the middleware components onto different heterogeneous devices in the environment. We envisage that by distributing the ubiquitous middleware into components on different devices would help us leverage the computationally strong devices, while keeping the energy preserve for the energy intensive devices. Further, the components of a middleware for ubiquitous systems vary widely in their subject orientation. We have specialized research areas and dedicated researchers for every component of the system e.g. context reasoning, service discovery, sensors and RFID are specialized research areas themselves. Therefore, by separating the middleware into modular services, these components could be developed separately and provided independently by different research experts. It would give the chance to manufacturers to improve or update their services independent of other services in the system; even they could charge for their services too.
In the extended domain name space figure, WOU stands for “A Web Of Ubiquitous Objects”. A question could arise here, why have we chosen the National ID-Card number or Social Security Number? The answer is two fold; we need a unique identification while keeping it comprehensible for a person. Almost, every person remembers his social security or National identification number. Although, initially it is difficult to remember, just like a phone number, but it has been observed that after using it few times, people tends to remember it.

This URI will be used to access it over the web and its properties can be manipulated via Http and XML as URI scheme and communication language respectively.

We envision that all the resources even humans should be accessed by a uniform identification mechanism [10], URI, so that we could have the uniform and interoperable accessibility mechanism and the existing world wide web could be leveraged to its full potential.

We provide the bird’s eye view of the system using above figure. It shows how could the use of URI for real world objects would benefit the existing architectures in the interoperation, creation of personal ontologies and in achieving the semantically ubiquitous web. It shows that an agent or user could access the World Wide Web, virtual resources by using URIs as well as the physical objects through the web.
2. Software Architecture

Based on our URI model, we propose a service oriented modular distributed software architecture, termed as ESAQ. An evolvable software architecture for ubiquitous systems. Our architecture follows a service oriented modular approach. By modular we mean that our software is distributed in components or modules and it is not necessary for each module to be on the same system, they could be distributed on different systems. e.g. A microwave oven does not need to possess a context reasoning mechanism, always, to make inferences. Only specified ontology represented in owl or RDF would be enough for microwave oven to keep with it. While, the context reasoning service, when needed, could be used, via URI, from some web server or any PDA etc, available in the ubiquitous environment. We employed http services in most of the devices (whose functionality need to be accessed via web), due to which those devices could be operated directly from the web. Further, if every person keeps his PDA/mobile phone with him then he would be able to manipulate the devices with his personnel configuration either in his own home or any other network [9]. This is manifested in the above figure.

We have represented different components of our software ESAQ, in the next figure. Some are mandatory while others are optional. The optional components depend on the device’s computational strength and need. This architecture could be deployed as a unit on single system or could pursue the modular approach. We prefer a modular approach to utilize the computational device. In case of a modular approach (as we proposed a URI for every device earlier), we’ll use the URI scheme to identify different components lying on different devices. Currently, for service discovery we are using UPnP in our system. Http services would be installed for devices which need to be accessed via web. With the http module, devices could be directly accessed via web, and monitored remotely.
To achieve the modularity, we distribute the components according to their tasks and needs. We'll briefly elaborate on each.

**URI Manager**: It is the main component which interacts with all the modules of the system. URI manager not only manages but also keeps record of the URIs of the components used. Particularly, in home environment, when the service is available on the near by device, it doesn’t use the internet name service, instead query the native service discovery protocol.

**URI Agent** module contains the two code mobility approaches, “Code On Demand” and “Mobile Agent”. We have employed Code on demand approach as well as mobile agent approach to achieve the code mobility[12], because a disadvantage of mobile agents is the need to install a support agent platform at each host the agent need to visit.

**Ontology Manager** module provides the mechanism to retrieve ontology for a particular service via URI Agent module and the methods to create, save and delete the ontology. It provides a user with the unique facility to create the ontology, and generates the schema for the ontology automatically. This ontology manager would not be needed by every device. Therefore, we would rather keep it in a separate device, like laptop or PDA or on the internet server, where it could be used by the user to create ontology and download the generated ontology.

**Service User Agent Interface** module acts as a presentation module, which interacts with URI agent to produce the interface for the users. This module would be appropriate for only the display devices, like Personal computer, TV etc.

**Context Reasoner** provides the reasoning mechanism on the ontologies provided by the stationary or dynamic context provider. By stationary we mean the devices which do not sense information like microwave oven, digital cameras, personal video cameras, but these devices contains their ontological schemas for their representation e.g. cooking recipe, personal schedule etc. By dynamic we mean the context gained from the sensors, RFIDs or some external web server to get the weather information. Context reasoning is a computation intensive process and it could cause bottleneck in pervasive systems. It would be better, if we would rather keep it to some computational intensive device and if needed it could be accessed by URI agent via URI.. URI agent could send the data to the device and download the result from it.
Context Converter: provides the mechanism for dynamic context providers to convert the sensed data to a context representation language such as OWL or RDF. This conversion allows the knowledge to be shared and reasoned according to a standard. Context converter as already mentioned, is suitable for dynamic context providers like surveillance video camera, sensors to detect human body temperature, pulse rate etc.

Repository: is a collection of the data present in the system. It keeps the data within the system e.g. Collection of service descriptions, ontologies and sensed data. Repository is of two kinds. First, one is a mini repository, which is kept by every device to keep the service descriptions and sensed data if any. Second one is a large repository which keeps the historical or unused sensed data for a time period specified by administrator. This type of repository is kept in a device, which enough or large disk space available for it. From time to time summarization technique can be applied to remove the already used data or deleted with the concern of administrator.

We are also integrating the OSGi based approach for our software, because it is evolving as a standard in ubiquitous environment for interoperability. This is shown in the above figure. Secondly, developing a separate service discovery interoperability middleware is a separate research area and is out of scope of our work and currently different service discovery consortium don’t agree on any common standard.

Some of the OSGi services include, administration, device management, HTTP and service discovery drivers for UPNP, JINI, Bluetooth etc. By using OSGi, it would be easy to remotely manage, update and deploy services for home users.
5 A Sample Scenario

Consider a person possessing digital mobile phone with MP3 player attached to it. He creates the personal ontology for his schedule in his laptop. He also creates the personal ontology for his digital mobile phone cum MP3 Player, via OntologyManager, to play songs according to his mood i.e. Schedule. This person uses the personal ontology for schedule by referencing his laptop.

Now, suppose this person moves to a party with his mobile phone. On entering the party hall, this mobile phone advertises its service on the local network along with the URI assigned to it. One another person’s Mobile phone in that party gets the notice of this service. The second Mobile+MP3 player already possess an ontology to play different songs, on the telephone calls from different persons (e.g. relatives, friends etc). The URI Agent in that mobile phone tries to use its own ontology with advertised mobile’s ontology to play songs as per the mood of user and the caller. For this to happen, it has to use Context Reasoner to get the new reasoned ontology. Since, context reasoning is computational intensive this mobile phone has to have the services of the context reasoner from some other device. If the Context reasoner is present in a local environment, the URI manager would use that otherwise it will try to utilize the service of some other Context Reasoner on the internet or some other environment by using the URLs to access that Context Reasoner. Finally, the reasoned ontology is used by the second mobile created by the use of Context Reasoner’s Reasoning Service.

This short and a simple scenario articulates the use of the proposed architecture in a very simple manner. A whole lot of other scenarios are possible, but to keep the proceedings simple and short we have mentioned this scenario only.
Conclusion and Future Work

• A service oriented modular distributed middleware has been proposed based on our Uniform identification of devices via URI model.

• We envisage that a modular middleware with URI approach would resemble object oriented paradigm with service oriented functionality.

Today, the researchers are trying to make the human intervention as less as possible in the computing. Our research is also focusing on this and this paper has proposed a service oriented modular distributed middleware, based on our proposed URI model for devices. We have also presented a sample scenario to explain our approach.

Currently, we are working on the implementation stage of our middleware. We are developing a model to simulate and analyze the results of the environment. Our approach follows inductive logic i.e. analysis of the models of specific environments then generalizing to a global environment. We are also thinking to employ OSGi not just as a gateway but to use OSGi framework in different computational devices, so that they could benefit from the OSGi functionality directly e.g. a personal device like mobile phone or PDA would be able to access different environments, running different discovery protocols and could be benefited by different OSGi functionality.

We envisage that a modular middleware with URI approach would resemble object oriented paradigm with service oriented functionality, just as in object oriented computing, researchers and engineers can focus only on their respective service, while leveraging the use of unified middleware. By resemblance with object oriented functionality we mean that the components of the system could evolve independently of other systems. Manufactures could provide their updates directly while the components interact with each other in service oriented manner.

Acknowledgement: This was supported by University ITRC Project of MIC.

References


[8] Pavlin Dobrev, AGDavid Famolari, Christian Kurzke, Brent A. Miller Device and Service Discovery in Home Networks with OSGi, IEEE Communications Magazine, August 2002


