Designing a Framework for Scalable Coordination of Wireless Sensor Networks, Context Information and Web Services

Yoshihiro Kawahara, Nao Kawanishi, Masahiro Ozawa, Hiroyuki Morikawa, and Tohru Asami
The University of Tokyo
{kawahara, asami}@akg.t.u-tokyo.ac.jp
{river24, ozawa, mori}@mlab.t.u-tokyo.ac.jp

Abstract

Context-aware services are one of the key applications in the ubiquitous computing environment where physical and virtual information are seamlessly integrated across the network. However, so far we have found few discussions on architectural styles for such integration of wireless sensor networks and other networks services and objects. Conventional context aware services have been built in an ad-hoc manner, making it difficult to build new applications or evolve existing ones. The objective of this paper is to discuss the architectural requirements from several points of view such as entry barrier, extensibility, interoperability, and scalability. Then we show a service coordination framework based on REST architectural style.

1. Introduction

The advent of ubiquity and miniaturization in computer networks has drastically influenced our views about computing platforms and virtual environments. In particular, wireless sensor networks will become an indispensable infrastructure as a nerve system in the future ubiquitous network society. Wireless sensor network (WSN) will capture real world context information about people and objects and make it possible to provide various kinds of helpful applications that are tied to our real daily lives[1].

Efforts having made for realizing practical sensor network ranging from hardware development to network protocols. However, in a ubiquitous network environment, not only do heterogeneous sensing devices need to be integrated, but also heterogeneous actuators and web information. So far we have found few discussions on architectural styles for such integration of WSN and other services and objects in ubiquitous network environment. Conventional context aware applications have been built in an ad-hoc manner, making it difficult to build new applications or evolve existing ones. At first glance, one might suggest that we deal with such integration in the same manner that we deal with different web services in WWW or different networked appliance in smart home environments. However, integration of various information sources with WSN is hard for several reasons. First the system needs to support for heterogeneous data types ranging from 1 bit flag of the RFID to video streams up to several hundred Mbps. Secondly, input data and output data themselves are provided by pervasive sources that are scattered over the Internet. Moreover, more and more network protocols and common API are created and never merged into one.

Thus, this paper aims at establishing a scalable networking architecture that seamlessly integrates heterogeneous information sources to provide rich context aware services. This paper focuses on discussion of architectural elements rather than implementation details. For more complete technical details see forthcoming papers.

2. Related Works

Classically the networking methodologies have been addressed within a development of middleware in the ubiquitous networking environment. There is a vast literature on the development of context aware applications.

A good recent example is the Context Toolkit which supports common features required by context aware applications such as capture and access of context from locally connected sensors, storage, distribution, and independent execution from applications[2]. Phidgets is also popular among researchers for prototyping purposes[3]. Phidgets provide both hardware modules and software libraries for creating robotics software applications. However these toolkits do not always suitable for providing scalable network applications over the Internet because their service coordination mechanisms are tied to specific programming languages.

Other approaches can be found in the standardization activities on generic service platforms and standard network protocols. OSGi has specified a Java-based service platform that can be remotely managed by defining an application life cycle model and a service registry[4]. UPnP supports for dynamic network configuration of home and office appliances based on IP-based communication standards[5]. DLNA is publishing network device interoperability guidelines for digital
contents to be recorded and played on consumer electronic devices[6]. Our paper rather focuses on developing a network architecture built on top of these standard protocols and frameworks.

3. Technical Challenges

In this section, we review software components for developing a typical context aware application in order to discuss architectural requirements. In short, a context aware application retrieves context information of a user and/or his environment and provides a service based on the context information. The service includes displaying some information, actuate real objects or robots in the environment, and send data over the network. Figure 1 outlines this procedure.

<table>
<thead>
<tr>
<th>Display</th>
<th>Actuators</th>
<th>Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Send Mail, Publish Blog, ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule Processing</th>
<th>Filtering</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Sensors</th>
<th>Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Blog, News, Live Video, ...</td>
</tr>
</tbody>
</table>

Figure 1. Typical Components of Context Aware Applications

3.1. Sensor data

Sensor data serves as an input data for context aware services. Typical sensor data includes temperature, brightness, and motion of human/animals. Location information derived from GPS and RFID tags can also be regarded as sensor information. In the typical wireless sensor networks, sensor data generated by each sensor node is routed toward a sink node. Then a system attached to the sink node processes the data format into appropriate data format. The sink node accepts a query from a client and serves as an interface between sensor network and applications. Existing standard sensor data description format include SensorML[7] and OSNAP[8].

3.2. Context Information

Context is defined as any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical or computational object. Context information is typically derived from sensor data generated from WSN and then fed into applications as input data. Higher-level context information is sometimes derived from lower level context information.

3.3. Information Sources on WWW

World Wide Web is an essential infrastructure for our daily lives. Until a decade ago, WWW was nothing but a huge information repository where knowledge is exchanged. Then after a dot-com era, it evolved into a virtual market in which people can take part in commercial activities without constrained by geographical limitation. After the emergence of Google, web has experienced the third evolution, which sometimes called Web2.0. The web services began to provide API for encouraging the WWW resources to be reused over the Internet. When resources of WWW are to be integrated with sensor information and context information derived by ubiquitous network infrastructures, we believe another revolution will occur.

3.4. Actuators and Human Interfaces

Actuators and human interface components act as means to provide services to the users. We do not have actuator networks whose architecture is fixed by the standard yet. One of the ongoing specifications for actuating robots over the network is CrossML[9].

4. Design Criteria

Taking all the components into consideration, we show design criteria for framework to coordinate different services in the ubiquitous networks environment.

4.1. Low entry barrier

Context aware applications are composed of heterogeneous resources as mentioned in the previous section. The number of created services grows proportional to the product of the number of available input resources (context) and number of available output resources (actuators, objects, etc). Therefore we need to incorporate a scheme to enhance the creation of components in an easy way. Technically, resource description scheme that allows rich description of heterogeneous resources is important. Secondly, it needs to support for a flexible resource identification mechanism that allows developers to easily create, modify and delete resources on the network and allows users to access the resources as simple as possible.
4.2. Extensibility

While simplicity makes it possible to deploy an initial implementation of a distributed system, extensibility allows us to avoid getting stuck forever with the limitations of what was deployed. Even if it were possible to build a context-aware system that perfectly matches the requirements of its users, those requirements will change over time, just as available context and information resources change.

4.3. Interoperability

As a context aware system is an ensemble consisting of different resources, we cannot necessarily assume that every resource generator agrees on the architecture we provide. For example we are experiencing that multiple heterogeneous network protocols running within a home network environment: some AV appliance talks over DLNA and others only accept a proprietary protocol. Moreover wireless sensor networks tend to be installed for dedicated purposes, incompatible with other home network infrastructures. Therefore architectural design should be made in the upper layers to absorb heterogeneity.

4.4. Scalability

Different from web and mail applications whose network session is rather short, context aware applications sometimes need to process a stream of information for a considerable time. (e.g. Continuous monitoring of temperature sensor data.) Moreover, some of sensor data are fed to multiple context inference engines and applications in order to organize multiple applications. That means information sources may encounter an unexpected number of requests from unknown clients. Therefore caching and load balancing mechanism should be supported as network architecture.

5. Architectural Style Based on REST

Based on the discussion mentioned in the previous section, we show a network architecture based on REST (REpresentation State Transfer)[10]. REST is an architectural style that emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulates legacy systems. WWW is one instance of a real distributed system based on REST. WWW enjoyed the scalability and growth as a result of a several design principles defined in REST. The design principles include:

- Application state and functionality are divided into resources
- Every resource is uniquely addressable
- All resources share a uniform interface for the transfer of state between client and resource
- A protocol that is Client/Server, Stateless, Cacheable and Layered

An important concept in REST is the existence of resources (sources of specific information), each of which can be referred to using a global identifier (a URI). In order to manipulate these resources, components of the network (clients and servers) communicate via a standardized interface (e.g. HTTP) and exchange representations of these resources (the actual documents conveying the information). Any number of connectors (e.g., clients, servers, caches, tunnels, etc.) can mediate the request, but each does so without "seeing past" its own request. If you apply this architectural style to the constitution of context-aware services, sensor information, context information, actuators and user interfaces correspond to resources.

With REST architecture, an application can interact with a resource by knowing two things: the identifier of the resource, and the action required – it does not need to know whether there are caches, proxies, gateways, firewalls, tunnels, or anything else between it and the server actually holding the information. In other words, the network can support for caching and load balancing without demanding applications to be modified. For this reason, this approach scales better than distributed object base technologies such as RPC and CORBA.

The application does, however, need to understand the format of the information (representation) returned, which is typically an HTML or XML document of some kind, although it may be an image or any other content.

6. CASTANET

In this section, we show design of a context-aware service coordination framework named CASTANET. The goal of CASTANET is to provide a unified method to access and coordinate pervasive resources with high extensibility, inter operability and scalability.

6.1. A Unified Interface for Pervasive Resources

The most important design principle in REST architectural style is resource access scheme. In CASTANET framework, sensor information, context information, web service resources, actuators and human interfaces are defined as resources and assigned unique URIs. Then clients are allowed to apply four http operations (GET, PUT, POST, and DELETE) on the URI. When clients GETs the URI, the client can download a current status of the corresponding resource. (The
Home appliance networks do not always support existing pervasive resources such as sensor networks and PUT/GET/DELETE/POST operations by clients. As provided on top of http server and allow server, client and resources. All the resources are etc.).

Weather forecast, stock quote, traffic alert information, to date information by Atom and/or RSS format (such as compatible web services, quite a few resources provide up Amazon E-Commerce Service (ECS). Except for REST Amazon.com provides their product data and e-commerce functionality by way of REST compatible API named Amazon E-Commerce Service (ECS). Except for REST compatible web services, quite a few resources provide up to date information by Atom and/or RSS format (such as weather forecast, stock quote, traffic alert information, etc.).

6.2. Resource Coordination Mechanisms

Figure 2 shows the relationship between CASTANET server, client and resources. All the resources are provided on top of http server and allow PUT/GET/DELETE/POST operations by clients. As existing pervasive resources such as sensor networks and home appliance networks do not always support for CASTANET compatible http interface, network administrator needs to install protocol converter in order to encapsulate existing systems.

A CASTANET client is an entity that coordinates various resources to compose context-aware services. The minimum requirement for CASTANET client is an ability to GET URIs and interpret the xml message generated by CASTANET server. Typical web browsers such as Internet Explorer, Firefox and Safari support for getting such resources. But when user wants to coordinate different resources over network, users can use open source software named Plagger. Plagger is a pluggable RSS/Atom feed aggregator written in Perl and capable of getting, processing, and publishing web based resources depending on the conditions.

CASTANET does not provide any strict definition of the semantics of the message but just provide recommended practice exploiting Atom format. Instead, each resource provider (CASTANET server) is requested to provide a reference to API documents inside the message. Other than API references, CASTANET server can provide a list of available resources on the server by predefined URI such as http://myhome.example.net/resources/.

The main reason we do not provide strict definition on the message format is to maximize the interoperability with existing web service resources. For instance, Amazon.com provides their product data and e-commerce functionality by way of REST compatible API named Amazon E-Commerce Service (ECS). Except for REST compatible web services, quite a few resources provide up to date information by Atom and/or RSS format (such as weather forecast, stock quote, traffic alert information, etc.).

6.3. AAA

Authentication mechanism between server and client bases on Web Services Security (WS-Security). The specification is flexible and designed to be used as the basis for securing Web. The algorithm uses WSSE Username Token, whose authentication is conducted as follows:

1. Start with 2 pieces of information: username and password.
2. Create a nonce, which is a cryptographically random string.
3. Create a "creation timestamp" of the current time, in W3DTF format.
4. Create a password digest by
   \[ \text{PasswordDigest} = \text{Base64} \left( \text{SHA1} \left( \text{Nonce + CreationTimestamp + Password} \right) \right) \]
This approach solves the password sniffing problem, and the replay attack.

7. Conclusion

In this paper, we discussed architectural requirements of a context-aware service coordination framework from several points of view including entry barrier, extensibility, interoperability, and scalability. Then we showed a design of CASTANET that is a service coordination framework based on REST architectural style. In CASTANET framework, sensor information, context information, web service resources, actuators and human interfaces are defined as resources and assigned unique URIs. Then the clients access to the resources with only four http operations (GET, PUT, POST, and DELETE). By using Plagger as a client, users can create complex context-aware application within this framework.

This service coordination framework is based upon REST architectural styles and thus inherently highly extensible, inter operable and scalable. Application services are composed by CASTANET clients in distributed manner and no central mechanism exists in this framework. Every CASTANET servers can support for load balancing and caching operation that are widely used in existing WWW systems. Therefore the CASTANET framework will scale over the Internet.

Currently we are developing the first version of CASTANET server to demonstrate our concept. Initial distribution supports for MICA motes, Phidgets, and Pavenet module as sensor and actuator resources.

References


