Towards Service-Oriented Middleware for Cyber-Physical Systems

Dat Dac Hoang[1] Hye-Young Paik[1] Chae-Kyu Kim[2]

[1] University of New South Wales, Australia
{ddhoang,hpaik}@cse.unsw.edu.au
[2] Electronics and Telecommunications Research Institute, Korea
kyu@etri.re.kr

Technical Report
UNSW-CSE-TR-201201
January 2012

THE UNIVERSITY OF
NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

We propose middleware, named WebMed, which is designed with a service-oriented view point to support Cyber-Physical Systems (CPS) applications. WebMed enables access to the underlying smart devices and integration of its device-specific functionality with other software services. It consists of five components: WebMed node, Web service enabler, service repository, engine, and application development. With WebMed, interacting with physical devices becomes as easy as invoking a computation service. Using the basics of service-oriented guidelines, we can build a loosely coupled infrastructure that exposes the functionality of physical devices to the Web for application development.

1 Introduction

Cyber-Physical Systems (CPS) are a new class of systems that tightly embed cyber capabilities in the physical world entities (e.g., humans, public transport, power grid, medical systems), to transform their interactions with the cyber world [4]. The recent developments in computing such as sensors, Radio Frequency Identification (RFID) and Near Field Communications (NFC) allow us to realise this type of systems where highly collaborative computations with real-time sensing, monitoring and management of change are required. CPS opens up new horizons for many applications (e.g., automated road and traffic control, effective energy consumption monitoring in buildings, ubiquitous healthcare and the like).

However, the design and realisation of the complex CPS applications are not easy. CPS brings about increasing challenges in supporting time-critical interactions, and managing large and complex context. We believe that it is important to have a solution for an agile, but dependable middleware that can support dynamically changing diverse requirements of CPS applications. The applications also should be easily built and deployed, perhaps even by technically inert people to suit their needs in-situ.

The research into middleware architectures or platforms for realising CPS applications is still in its infancy. As we will discuss in the related work, there are some recent proposals that indicate the high level of interests in combining the Web architectures with CPS. Considering the success of the Web as the largest distributed system ever built, the new generation middleware architectures such as Web-of-Things, or Web service oriented paradigms are likely candidates to enable the connection between physical things in CPS and the Web in the cyber and human worlds.

In this paper, as an early design of our on-going work, we propose a Web service-based middleware architecture which aims to bring the service orientation paradigms into CPS architecture design. Service orientation leads to a standardised and unified infrastructure, built over the Web, in supporting of the utilisation of physical devices, computing elements and other software services together.

In our design, we aim to address the following issue: physical devices are highly proprietary in nature that it is difficult to create a connected environment containing heterogeneous devices (i.e., accesses to devices are rather tightly-coupled). Even when they can connect to each other, the capability of combining the data from multiple devices and their functionalities to create an ad-hoc application is very limited. Furthermore, the capability of integrating data and functionalities of physical devices with non-physical data and functionalities (e.g., software services) is also limited.

Our architecture design pays particular attention to bringing the underlying physical devices' capabilities directly to the application development layer, and linking them with non-physical device services, using service orientation principles such as loose-coupling, repository and discovery, reuse, and composition of services.

2 WebMed Architecture

WebMed aims at facilitating the service-oriented architecture for physical devices. The middleware contains high-level, logical representations of physical devices, computing elements and software services that are not necessarily linked to physical devices. WebMed caters for three different types of users: administrators, service developers and end-users (i.e., application users).

WebMed consists of five components: device adapter (named WebMed node), Web service enabler, service repository, engine, and application development (see Figure 2.1). The figure also shows the components in the physical layer which consists of physical devices and intelligent devices (i.e., these devices are equipped with software that enables remote control/access to the devices). In this paper, we do not focus on the physical layer of the CPS architecture. We rely on the existing work for providing the solution for dealing with physical devices or intelligent devices (e.g., their device drivers and device's API). In the following, we outline the main functionalities of each component.

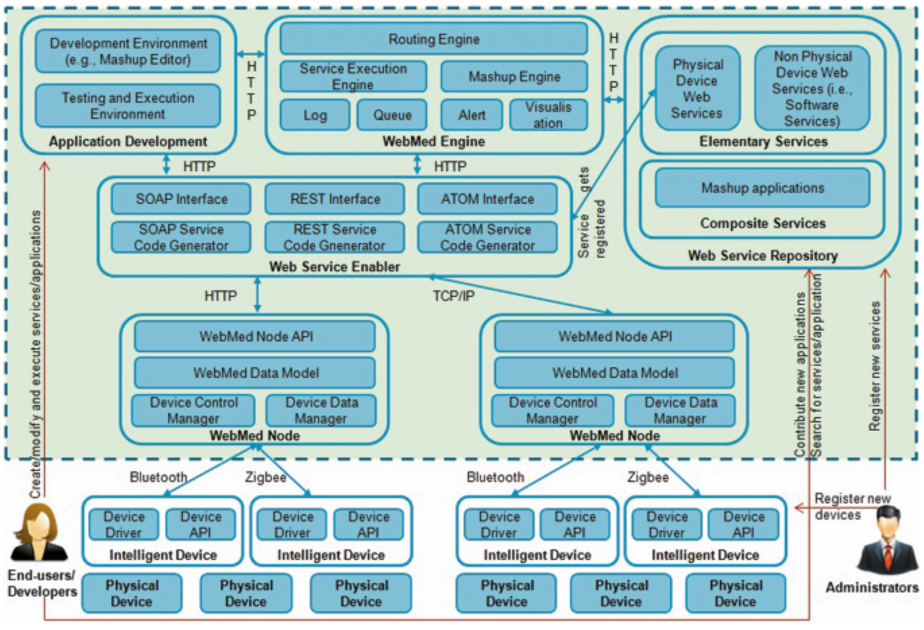

Figure 2.1: The WebMed middleware

2.1 WebMed Node: Device Adapter

A WebMed node acts as an adapter and device aggregator that "standardises" the heterogeneous devices' hardware, data structures, communication protocols and device control issues. It is responsible for consolidating underlying devices' data into a common model and controlling devices through the proprietary device drivers and APIs. It is the point of entry for the devices to our middleware.

Plug-and-Play

A WebMed node enables a plug-and-play environment for adding/removing newly connected/disconnected physical devices to the middleware. Once a device is connected to WebMed, the administrator needs to register and configure it so that the other components can recognise it as a new component. The administrator needs to upload and link new device with its driver (e.g., dynamic link library or java class) through an interface. The "device control manager" module generates proxy code for operations provided by the driver (e.g., by using the *WSDL* tool in Microsoft .NET SDK framework) which are exposed through WebMed Node control API. Since devices only provide raw data, the "device data manager" module processes (e.g., cleans, transforms, aggregates, filters) the raw data and creates a device's data table in the consolidated database in each node. All of the data here then be exposed as data-querying services through WebMed Node data API, and then by the Web service enabler.

WebMed Data Model

WebMed node contains a common data model element that provides uniform access to its underlying data from devices. Amongst the data managed by the node, there is the temporal and spatial information of physical devices, which provides the critical context information for CPS applications. The administrator can use either WGS-84 GPS code (e.g., latitude, longitude, altitude) or hierarchical naming model to manage the geographical information of originating devices. For example, an identification *Bondi.G1.L2.A5* represents for a parking slot in *Bondi Junction* shopping centre garage, garage building *1*, level *2*, row *A*, slot number *5*.

Networking Protocol

WebMed node utilises various technologies such as RFID, NFC and wireless sensor network (e.g., ZigBee, Bluetooth, 6LoWPAN) to communicate with the device over a network.

2.2 Web Service Enabler

Web service enabler provides a mechanism for the data and functionality of the physical devices to be accessible as Web services, that is it is responsible for service-enabling the devices. The core elements of Web service enabler are code generators and a Web server which provides a hosting environment for all Web service created by the enabler.

Web Service Interfaces

There are three types of Web services supported in the architecture: REST services, SOAP-based services and ATOM-based data feed services.

Service Code Generators

Based on the proxy code created by the WebMed nodes, this component generates matching Web services (e.g., using C# compiler application). The database managed by “device data manager” is exposed as REST services, as is the control functionality managed by “device control manager” (i.e., turning a device on). The same (or part of) operations of a WebMed node can also be exposed as SOAP-based services or ATOM feed services.

Operations

Web service enabler is an intermediate gateway between physical devices and cyber/computing elements. It plays a role as a stub for remote devices performs following tasks: (i) Receives a request from caller; (ii) Initiates the connection with WebMed nodes; (iii) Invokes proper operations provided by the device data manager and control manager; (iv) Waits for the result of the invocation; (v) Returns the value or exception to the caller.

Transportation Protocol

The enabler relies on TCP/IP and HTTP protocols as means for transporting messages.

2.3 Service Repository

Service repository contains two main types of services: elementary and composite. The elementary services are again divided into two categories: Web service for interacting with physical devices (what we would refer to as “physical services”), and any other services (what we would refer to as “software services”). The physical services are generated by the Web service enabler and registered in the repository. The repository categorises them by the meta-data relating to the spatial and temporal attributes of physical devices. Non-physical Web services are other Web services (i.e., that are not generated to control physical devices). A composite service is created by combining elementary services. A lightweight composite service (i.e., mashups) is also considered as it allows easier creation and sharing of applications amongst the end users. The CPS application developers or the end users can add more composite services to the repository, although the elementary services are managed by the administrator.

2.4 WebMed Engine

WebMed engine is the core element providing a runtime environment for all Web services and operations in the middleware. WebMed engine uses HTTP as transportation protocol and contains the following modules:

Routing Engine

It takes care of processing requests from the WebMed application component and dispatches them to the right evaluation engine. For example, invocation request of a single Web service should go to “execution engine” module and execution request of a mashup application should go to “mashup engine”. Routing engine also connects to “queue service” to control messages.

Execution Engine

It is responsible for invoking services. It also has a comprehensive locking, provenance and data transfer model that allows multiple service invocations to run at the same time.

Mashup Engine

It enables a mashup execution by resolving the integration logic between services. Mashup integration logic can be defined within the request-response interaction fashion and using pipeline mechanism. Also, there is an event management service to manage the control flow between services.

Alert Server

It utilises the publish/subscribe paradigm enabling users to express their interest (i.e., subscribe) in certain kind of events and subsequently are notified by the server (i.e., publish). This module produces data in the format of RSS/Atom feeds.

Logging Service

It records all system's and users' activities of the middleware.

Queue Service

It stores and forwards messages from routing engine to "execution engine" or "mashup engine". It also contains a persistent storage to store messages and data. This module is used to avoid collision (e.g., two invocation requests to operate a physical device at the same time).

Visualisation Module

It is responsible for rendering invocation/execution results on execution environment. The environment for visualisation is a Web browser.

Monitoring Server

It allows the administrator/developers to track message flows and detect errors in the execution.

2.5 WebMed Application

The WebMed application component provides high-level management of interaction and composition of Web service components in the middleware. This component serves as user interfaces for developers and end users to invoke a Web service, to create a mashup application and composite services, to monitor a physical device, or to subscribe for alerting service of a Web service. To use the services, users have to authenticate themselves to get access level and personal settings. User access level (e.g., level 1 can only gets data from devices and cannot control devices) is granted by administrator and personal settings (e.g., login detail, interested device) is maintained by users.

There are two modules in this component: development and testing/execution environments. Developers and end users use development environment to create applications. They can combine the functionality of a physical service with other computing/software service or even with mashup applications. The application is then deployed and visualised in the execution environment. The execution environment can also be used as a testing environment before final deployment.

2.6 Implementation

Accordingly to the architecture described above, we intend to implement the middleware on top of the Apache Axis framework running on Jakarta Tomcat Web server. Database manager is implemented by using TinyDB [3]. We rely on RESTlet[1], RSS.NET[2] frameworks to implement control manager. The language for implementation is Java and C#.

3 Car Park Management Scenario

To show case an application of WebMed middleware, let us consider the following scenario. A big shopping centre chain who owns car park buildings in multiple locations commissioned WebMed middleware to be installed in all car park buildings. Over the years, sensors were installed in each car park slots, but the different device types and manufacturers were used. WebMed creates several WebMed nodes fitted to individual location to build a middleware layer that hides the underlying heterogeneity.

Being able to use the repository which gives access to physical services and software services enables the administrator to manage and control car parks efficiently (e.g., instantly knowing how many spots are occupied at any given time, showing customers the shortest route to an available parking slot). In addition, the ability to integrate functionality of physical device and software services facilitates new value added services to customers (e.g., reserve a parking slot, pay for parking ticket using mobile device, SMS alert when overtime).

Figure 3.1 provides a high-level view of WebMed operations from the end users' and application developers' perspectives. In the application level, there are some applications such as reserve a parking slot; send a short message to customer's mobile number when the parking permit is running out; send an alarm to the car park administrator when there is a car left behind after closing time. These applications consume existing Web services and/or mashup applications in the WebMed service repository. For example, there are physical services to control parking sensors such as turn on/off a sensor (or a group of sensors), and check for availability of a parking slot, as well as software services such as currency conversion, payment and send SMS Web services.

Let us say that a user wants WebMed to send an SMS to her mobile when her reserved parking slot becomes available earlier than the arranged time (e.g., the previous car left the spot early). If such an application is not already available in the repository, she can create one by combining "Availability" Web service (a physical service) which will sense the parking slot becoming available,

[1]http://www.restlet.org/

[2]http://www.rssdotnet.com/

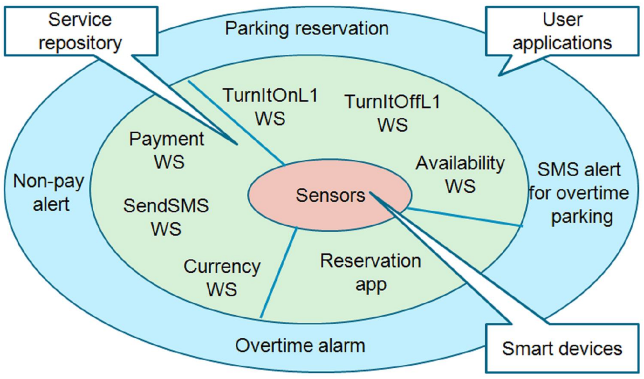

Figure 3.1: Garage management use case

“SendSMS” service (a software service) which sends the message, and parking reservation application (an existing mashup application) which allows her to complete the parking and pay process.

4 Related Work and Conclusion

As mentioned, there are active interests in the topic of middleware design that combines Web and CPS. Zhang et al. [7] proposed a reconfigurable real-time middleware for distributed CPS with aperiodic event. However they only concentrate on physical layer of the architecture. Dillon et al. propose a framework to integrate Web-of-Things and CPS [1]. Guinard et al. [2, 6] proposed how an actual Web server can be implemented on tiny embedded devices to turn them into RESTful resources. When computational resources are too limited or devices do not offer a RESTful interface, they propose to use an intermediate gateway that can offer a unified REST API to access these devices, by hiding the actual communication protocols used to interact with them. Our work differs from these works in that we support not only REST but also SOAP and Atom feeds. This diversifies the service interfaces of physical devices, thus make users more flexibility in integrating device’s functionality with others. We also consider a repository component to increase support for the application development and reuse of existing services. Stirbu [5] proposed to use RESTful principles to provide a Web of Things framework. However, they mainly focus on device discovery (i.e., not the functionality offered by devices).

In this paper, we presented our initial WebMed design and implementation plan. Immediate future work is on the WebMed prototype implementation and improvements, and use case scenarios in different application domains. We will evaluate the benefits and limitations of WebMed through these scenarios.

Acknowledgments

This work was supported by the IT R&D Program of MKE/KEIT [10035708, “The Development of CPS (Cyber-Physical Systems) Core Technologies for High Confidential Autonomic Control Software”].

Bibliography

[1] Tharam S. Dillon, Hai Zhuge, Chen Wu, Jaipal Singh, and Elizabeth Chang. Web-of-things Framework for Cyber-physical Systems. *Concurrency and Computation: Practice & Experience*, 23:905–923, June 2011.

[2] Dominique Guinard, Vlad Trifa, and Erik Wilde. A Resource Oriented Architecture for the Web of Things. In *Proceedings of the 2010 conference on Internet of Things (IOT'10), Tokyo, Japan*, 2010.

[3] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: An Acquisitional Query Processing System for Sensor Networks. *ACM Transactions on Database Systems*, 30:122–173, March 2005.

[4] Radha Poovendran. Cyber-Physical Systems: Close Encounters Between Two Parallel Worlds. In *Proceedings of the IEEE*, volume 98, pages 1363–1366, August 2010.

[5] Vlad Stirbu. Towards a RESTful Plug and Play Experience in the Web of Things. In *Proceedings of the 2nd IEEE International Conference on Semantic Computing (ICSC2008), Santa Clara, CA, USA*, pages 512–517, 2008.

[6] Vlad Trifa, Samuel Wieland, Dominique Guinard, and Thomas Michael Bohnert. Design and Implementation of a Gateway for Web-based Interaction and Management of Embedded Devices. In *Proceedings of the 2nd International Workshop on Sensor Network Engineering (IWSNE'09), Marina Del Rey, CA, USA*, 2009.

[7] Yuanfang Zhang, Christopher Gill, and Chenyang Lu. Reconfigurable Real-Time Middleware for Distributed Cyber-Physical Systems with Aperiodic Events. In *Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS '08), Beijing, China*, pages 581–588, 2008.