Service-Oriented Middleware Architectures for Cyber-Physical Systems

Dat Dac Hoang, The University of New South Wales Sydney, NSW 2052, Australia

Hye-Young Paik The University of New South Wales Sydney, NSW 2052, Australia

Chae-Kyu Kim, Electronics and Telecommunications Research Institute Daejeon, Korea

Summary
Cyber-Physical Systems (CPS) roughly refer to the integration of computation (cyber) and physical processes. In CPS, embedded devices/computers and various sensors are interconnected to monitor and control the physical processes. The recent interests research into CPS highlighted some of the design challenges in building CPS and the need for broader research into middleware architectures. In this paper, we first present systematic literature survey of research outputs in CPS middleware designs to (i) present the state-of-the-art and (ii) to bring out some research focus on the issue. Then, we propose our early conceptual middleware design named WebMed. The middleware is designed with a service-oriented view point to support 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.

Key words:
Cyber-Physical Systems, Service Oriented Architectures, Literature Review, Middleware Design

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 [1]. 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.

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. However, the research into middleware architectures or platforms for realising CPS applications is still in its infancy. While some approaches enable physical devices 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. More importantly, their complexity and lack of standards lead them to a rather a small community of developers and technical users hence their daily life usage has been limited.

In this paper, we first set out to do a systematic literature survey on Cyber-Physical Systems with a particular focus on middleware and architecture designs. Then, 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

Manuscript received January 5, 2012
Manuscript revised January 20, 2012

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.
The remainder of this paper is organized as follows. In Sections 2 and 3, we present a systematic literature review on realising service-oriented architecture for CPS design. In Section 4, we present WebMed, a service-oriented middleware for CPS. We conclude the paper and present the future work in Section 5.

2. Systematic Literature Review

For the review of research activities in the area, we present a systematic literature review on realising service-oriented architecture for CPS design. We followed a methodology proposed by Prof. Kitchenham [2] which provides a set of guidelines for software engineering researchers for producing a literature survey that is a fair evaluation of a research topic by using a trustworthy, rigorous, and auditable methodology. Hence, the structure of this section is to explain the process of collecting the evidence of the literature data, and then summarise the trend of CPS research papers as an overview of the research activities. We then, particularly focus on summarizing papers on the service-oriented architecture aspects in the CPS middleware design. We present the findings in the following section.
Service orientation paradigms lead 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. We hoped that the review will be useful for the researchers and developers in the area to understand the landscape of service-oriented CPS, the strengths and weaknesses of the current approaches, and allow us to develop initial ideas on creating more advanced middleware.

2.1 Review Question

The scope of research issues in CPS is large, ranging from the areas in computer hardware and sensors/devices to software and complex architectures. As more researchers become involved in this "hot" area, the issues become dynamic and rapidly changing, which has influenced the scope of the study. Therefore, we need to specify research questions to be used in the review in a manner that allows us to limit the scope of the study. The research question for the review is framed as follows.
"*Consider CPS as an information system, what are the architectural styles needed to consider when we design and implement middleware architecture enabling a service-oriented CPS?*"

2.2 Data Collection Process

2.2.1 Search Criteria

In the first stage, we searched papers, research reports, books, dissertations and documents for the review. The key term "*Cyber Physical Systems*" was used to input into the Google Scholar [3] search engine. No limitation was set on the published time of the publications. The search results were saved and processed in the selection stage.

2.2.2 Selection Criteria

The selection process was conducted as in the following steps:

- Primary search: Papers were selected if they included the key term in the title. At this stage, we did not consider the quality criteria of the papers (i.e., all papers satisfies the key term were selected for the preliminary list).
- Preliminary list: The preliminary list was then filtered based on the following criteria: (i) publication is published from computer science venues (conference, journal or computer science organisation), (ii) the language for publication is English, (iii) publication focuses on CPS research, (iv) publication is available in full-text. We did not have any bias towards author name, author's institutional organisation or author's country. We, then, removed the duplicated search results due to published in difference online databases (e.g., ACM digital library, IEEE Explorer). We downloaded and saved the full-text of selected publications.
- Screening: In the screening step, full-texts of the remaining publications were assessed. Since this review focused on the middleware that enabling the service-oriented capability of CPS, we excluded papers that (i) are not relevant to the review question, (ii) are not thoroughly reviewed by reviewers.

After the screening step, we ended up with a list of papers for in-depth study.

2.2.3 Data Extraction

After the selection process, data extraction was used to draw out key themes as part of the synthesis stage of the review. We extracted the following information from the full-texts: authors, author's institutional organisation, author's country, year of publication, publication venue, publication's focus, and publication's result.

We thoroughly reviewed the publications in term of their architectural styles.

2.3 Synthesis

Table 1 shows the results of the search and selection steps in the process. A total of 2,750 publications were identified in the primary search step. In the preliminary list step, 399 publications were selected as they came from computer science venues. Five and eleven of which turned out to be not in English and duplicated, respectively, then were eliminated. We could not retrieved full-text for 57 out of 383 publications. After full-text screening step, 4 papers were excluded since they were not peer-reviewed publications. We also eliminated 264 papers from the list since they were not relevant to the research question. We classified the publications into four categories: Design, Security, Software Engineering, and Survey. We only selected publications from the "Design" category with their main research focuses on high-level design of the middleware for CPS (i.e., not in physical design of the CPS elements). Finally, 58 full-texts of publications were chosen for the in-depth study process of the review. Figure 1 shows the number of publications by category in our review list.

Table 1 Selection Synthesis (The numbers are as of 04 Jan 2012, by Google Scholar)

Step		Elimination	Result
Primary Search		-	2,750
Preliminary list	CSE venue	2,351	399
	English only	5	394
	Duplicated	11	383
	Full-text	57	326
Screening	Reviewed	4	322
	Relevant	264	**58**

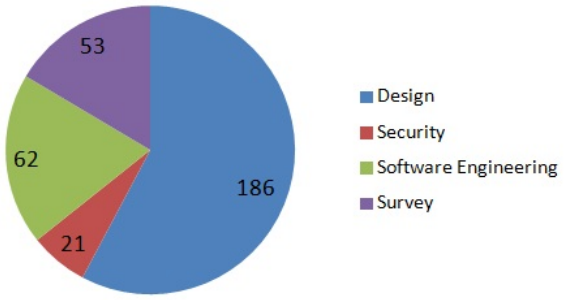

Figure 1 Number of Publications by Category

A majority of publications (186) is on design (e.g., modeling, designing and developing) of CPS. A large number of studies (62) focused on software engineering aspects of the CPS (e.g., test, QoS, performance, reliability, fault, resilient, privacy, evaluation). Also, a large number of papers (53) worked on surveying CPS including survey, taxonomy, classification, introduction, comparison, case study, position paper, point of view, and analysis. Only a small number of publications (21) concentrated on security issues of CPS. The published years of selected publications were in the period of 2006-2012.

3. Survey Findings and Discussions

3.1 General Architecture of Service-Oriented CPS

Unlike conventional embedded systems with concentration on the physical devices, service-oriented CPS is typically designed to coordinate the computational and physical parts of a system. Since physical devices typically have limited resources and computation power, it is not always possible to design and execute complex computation and processes. To cope with this challenge, some approaches propose a service-oriented architecture of CPS to handle complex processes using computational and physical resources. The benefits of using service-oriented architecture are twofold: (i) provide users a unified access/communication protocol to interact with physical and software elements, (ii) facilitate to build flexible systems while preserving efficiency and scalability.
In general, in the reviewed works, the generic architecture of service-oriented CPS contains three layers: access, service, and application.

- ***Access layer***: facilitates a standard platform for physical devices. First, it provides a networking environment for connecting devices (e.g., RFID, NFC or wireless connection such as IEEE 802.15.4, ZigBee, 6LoWPAN, or Wireless M-Bus). Second, it provides a common representation of physical devices in the system. The data model is used to accommodate various devices and resource data formats. The meta-data relating to physical devices are also included such as spatial attribute, temporal attribute, state representation. Third, it has responsible for device management by detecting and identifying newly connected/disconnected physical devices and their resources.
- ***Service layer***: is a logical abstraction layer on top of the access layer. The main goal of this layer is to cope with the heterogeneity issues of different physical components provided by the access layer and software components. Typically, this layer relies on the Web protocol such as TCP/IP, HTTP. First, service layer provides mechanisms (e.g., adapter, wrapper) allowing to transform physical device to physical service component. It defines the data retrieval for services and device access methods (e.g., push, pull, publish,

subscribe). Second, it allows discovering and registering new software components (e.g., computational services). Third, it also provides a common repository of component services to be used in the system. The service repository maintains comprehensive information about published services in term of functionality, QoS, and context information (e.g., sensor, actuator, temporal, spatial, and device related information). Fourth, it also concerns the QoS issue of the provided services. It ensures the adaptability, composability of services so that the users can compose applications in the application layer.

- ***Application layer:*** provides high-level management and interaction with physical and software components. It allows users to create applications with ease as create Web 2.0 mashups (i.e., lightweight, ad-hoc composition). It also offers other features such as searchability, sharing, composition, reuse of created application. Typically, the application layer contains a development environment and a runtime environment for creating and executing applications, respectively.

Table 2 Mapping of the architectural layers of reviewed work with generic service-oriented CPS design

Paper	Architectural layers		
	Access	Service	Application
[8]	Environmental Tier	Control Tier	Service Tier
[4]	WoT Device WoT Kernel	WoT Overlay WoT Context	WoT API
[9]	Network Layer	Dynamic Control Middleware	Application Layer
[6]	Local Resource Manager	Global Resource Manager	Qbroker
[14]	Service Provider Tier	Gateway Tier Negotiator	Application Tier
[7]	Physical Layer	Service Layer	Application Layer
[5]	Access Layer Federation Layer Provision Layer	QoS/QoD Enforcement Layer Event Service	Application Layer

A majority of the publications (31) have generic functional designs of their service-oriented CPS (e.g., [4-9]), while four focus on domain specific design of CPS (e.g., water distribution [10], home control [7], energy management [11], video communication between 3G phones and Internet hosts [12]). A majority of the publications concentrate on theoretical design of the CPS (e.g., [4, 8, 13]) while only six publications mention the implementations of their CPS (e.g., [9, 14-16])

In Table 2, we show how to map the architectural designs of the reviewed works into the generic architecture of the service-oriented CPS.

Depending on the functionality described by each layer, we categorised it to the layer of the generic architecture. For example, in [8], the "Environmental Tier" consists of physical devices and an environment for interacting with these devices so that we recognised it as the "Access Layer". The "Control Tier" controls devices, receives/monitors the data and let services invoke the physical devices so that we classify it as "Service Layer". Since the "Service Tier" contains a computing environment with services in SOA and manages reusable services, we put it into the "Application Layer".

3.1.1 Architectural Topology

We identified three topologies of reviewed works:

- ***Centralised***. In this topology, a middleware is deployed in a central server while collecting data and/or controlling sensors/actuators connecting to physical devices in distributed areas. For example, Llama [6], Bundle [14], AnySense [12], [17], [8], [18], [7], [5], [4]. The benefit of the centralised topology are (i) It makes easier for the management of CPS, (ii) It provides more secure environment. However, as the system complexity increases (e.g., the number of physical devices), this tightly coupled topology becomes problematic.

- ***Distributed***. In a distributed topology, each physical device implements a tiny "middleware" to control the physical part and connects with other in peer-to-peer model. For example, Iqbal and Lim [10] propose a middleware containing entities called agents and actors. These entities move across the network of sensor nodes facilitating the adaptive load balancing, monitoring, and activating resources. Lin et al. [19] views and models CPS as a multi-agent system, where each local agent communicates with other agents. Kim et al. [15] includes a network of cyber-nodes that provide computing resources. Each cyber-node has its own knowledge base and database to be shared among the network. This topology encourages scalability of the system as the computing elements and physical devices can be incrementally added without little interference with other elements. Also, distributed topology has no bottle neck point in the architecture so that it can minimise the network congestion. However, since these physical devices have limited resources, the limitation of this topology is that it is not possible to execute complex computation and processes. In addition, the tasks of devices registration, service discovery become harder to manage.

- ***Nested***. In the nested topology, physical things in CPS are deployed both in centralised and distributed style. The CPS may include one or more local CPS networks [9].

3.1.2 Data model

In CPS, where numerous physical devices connect, process, and exchange information to each other, a common understanding for all devices is crucial. This leads to a need to have a common data model accommodating physical things and software artifacts. This data model will be used as an integral part of the infrastructure facilitating an agile environment with minimal coupling between all its components. To model physical elements and software artifacts of CPS, we need to consider functional properties (e.g., energy, memory), non-functional properties (e.g., safety, reliability, latency, throughput), and interaction between them (e.g., event). In the reviewed work, we identified two data models used in CPS:

- ***Web-based data model.*** In this model, physical devices, software services and data produced by them are represented as Web resources. Interaction with component can be done in request/response, publish/subscribe styles. For example, users send command to turn on a device, read temperature information, or subscribe to get update information about the device's status. This model and interaction style is commonly used in the reviewed works. There are several works based on a wide adopted ontology model OWL-S (e.g., [13],[19]). For example, Huang et al. [13] define PE-ontology model to describe physical entities and services. Retrieving information from the components, it maintains a hierarchical meta-data structure including three main parts: ServiceProfile, ServiceModel, and ServiceGrounding. Park et al. [9] utilises IPv6 to enable the interaction between control devices and control device managers. The data model used by control devices and control device manager is XML-based data model.
- ***Event-based data model***. In this model, data and events generated by physical devices and software components are used to define consequent activities in the workflow. We pointed out two modes of event-based data model: periodically and sporadically. In the periodically mode (pull-based), data is read at regular intervals and analysed on-the-fly. This mode is commonly used for monitoring scenarios (e.g., read the sensor status for every 5 minutes or get humidity in the room in every minute). In the sporadically mode (push-based), a device emits an event whenever predefined conditions are met (e.g., send an SMS notification as soon as the humidity level raises above 90 percent, or alert administrator as soon as a camera detects movement in the room). In [10], the data model of components includes events. Each event and corresponding actions are critical to decide which followed components must have an action so that the required task is done in the correct context and under given constrains. Tan and Goddard [20] propose a Spatio-Temporal Event Model for CPS. It identifies the close interaction between cyber and physical world in both time and space and defines two event models: time model and spatial model.

4. SOA-based CPS middleware: WebMed

In this section, we sketch the design of our middleware named WebMed. As mentioned earlier, we believe that it is important to have a middleware solution for an agile, but dependable middleware with compose-able and reusable components that can support dynamically changing diverse requirements of CPS applications. More importantly, The applications also should be easily built and deployed, perhaps even by technically inert people to suit their needs in-situ.

Considering the success of the Web as the largest distributed system ever built, the new generation middleware architectures such as Web-of-Things, or service oriented paradigms are likely candidates for CPS.

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). 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 work, 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.

4.1 WebMed Architecture: Overview

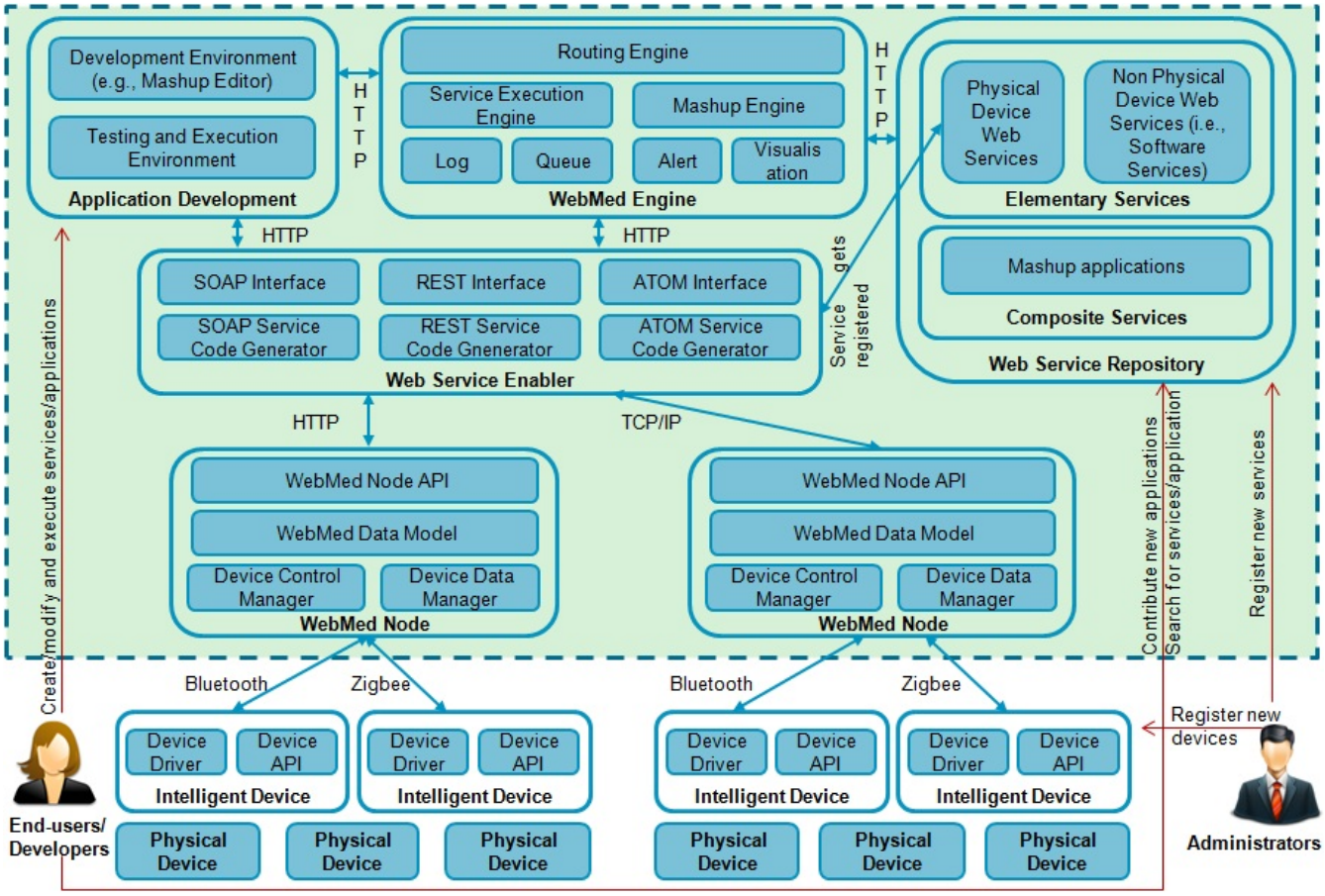

Figure 2 WebMed Architecture Overview

4.1.1 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.

4.1.2 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.

4.1.3 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.

4.1.4 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.

4.1.5 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.

4.2 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 [21]. We rely on RESTlet [22], RSS.NET [23] frameworks to implement control manager. The language for implementation is Java and C#.

4.3 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 2 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, "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.

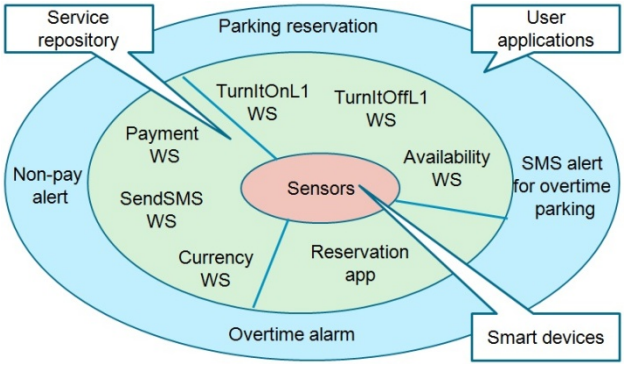

Figure 2 Garage Management Use Case

5. Conclusion and future work

In this paper, we presented a literature review on service-oriented middleware architectures for CPS. We analysed generic architecture designs, architectural topology, component data model. We then proposed WebMed, our early design of service oriented CPS middleware for ad-hoc applications. 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"]

References

[1] Poovendran, R., Cyber-Physical Systems: Close Encounters Between Two Parallel Worlds. Proceedings of the IEEE, 2010. 98: p. 1363-1366.
[2] Kitchenham, B., et al., Systematic Literature Reviews in Software Engineering - A Systematic Literature Review. Information and Software Technology, 2009. 51: p. 7-15.
[3] Google. Google Scholar Search Engine. 04-Jan-2012]; Available from: http://scholar.google.com.au/.
[4] Dillon, T.S., et al., Web-of-things Framework for Cyber-physical Systems. Concurrency and Computation: Practice & Experience, 2011. 23: p. 905-923.
[5] Kang, W. and S.H. Son, The Design of an Open Data Service Architecture for Cyber-Physical Systems. ACM SIGBED Review, 2008. 5: p. 3:1-3:2.
[6] Lin, K.-J. and M. Panahi. A Real-time Service-Oriented Framework to Support Sustainable Cyber-Physical Systems. in Proceedings of INDIN '10. 2010. Osaka, Japan.
[7] Lai, C.-F., et al., OSGi-based Services Architecture for Cyber-Physical Home Control Systems. Computer Communications, 2011. 34: p. 184-191.
[8] La, H.J. and S.D. Kim. A Service-Based Approach to Designing Cyber Physical Systems. in Proceedings of ICIS '10. 2010. Kaminoyama, Japan.
[9] Park, S.O., et al., A Dynamic Control Middleware for Cyber Physical Systems on an IPv6-based Global Network. International Journal of Communication Systems, 2011.
[10] Iqbal, M. and H.B. Lim. A Cyber-Physical Middleware Framework for Continuous Monitoring of Water Distribution Systems. in Proceedings of SenSys '09. 2009. Berkeley, CA, USA: ACM.
[11] Parolini, L., et al. A Cyber-Physical Systems Approach to Energy Management in Data Centers. in Proceedings of the ICCPS '10. 2010. Stockholm, Sweden.
[12] Xing, G., et al. Toward ubiquitous Video-based Cyber-Physical Systems. in Proceedings of SMC 2008. 2008. Singapore.
[13] Huang, H.-M., et al. Cyber-physical Systems for Real-time Hybrid Structural Testing: a Case Study. in Proceedings of ICCPS 2010 2010. Stockholm, Sweden.
[14] Vicaire, P.A., et al. Bundle: A Group based Programming Abstraction for Cyber Physical Systems. in Proceedings of ICCPS '10. 2010. Stockholm, Sweden: ACM.
[15] Kim, M., et al. An Application Framework for Loosely Coupled Networked Cyber-Physical Systems. in Proceedings of EUC '10. 2010. Hong Kong, China.
[16] Basanta-Val, P., M. Garcí anda-Valls, and I. Esté andvez-Ayres. Towards a Cyber-Physical Architecture for Industrial Systems via Real-Time Java Technology. in Proceedings of CIT 2010. 2010. Bradford, UK.
[17] Tan, Y., S. Goddard, and L.C. P\'erez, A Prototype Architecture for Cyber-Physical Systems. ACM SIGBED Review, 2008. 5: p. 26:1-26:2.
[18] Park, M.J., et al. Dynamic Software Updates in Cyber-Physical Systems. in Proceedings of ICTC '10. 2010. Jeju Island, Korea.
[19] Lin, J., S. Sedigh, and A.R. Hurson. An Agent-Based Approach to Reconciling Data Heterogeneity in Cyber-Physical Systems. in Proceedings of IPDPSW '11. 2011. Anchorage, AK, USA: IEEE.
[20] Tan, Y., M.C. Vuran, and S. Goddard. Spatio-Temporal Event Model for Cyber-Physical Systems. in Proceedings of ICDCS Workshops '09. 2009. Montreal, Canada.
[21] Madden, S.R., et al., TinyDB: An Acquisitional Query Processing System for Sensor Networks. ACM Transactions on Database Systems, 2005. 30(1): p. 122-173.
[22] Noelios. Restlet - RESTful web framework for Java. 04-Jan-2012]; Available from: http://www.restlet.org/.
[23] ToolButton. RSS.NET: An Open-source .NET Class Library for RSS Feeds. 04-Jan-2012]; Available from: http://www.rssdotnet.com/.

Dat Dac Hoang is a PhD candidate at the school of Computer Science and Engineering, University of New South Wales, Sydney, Australia. His PhD topic is on designing a mashup architecture on spreadsheets. His recent research interests include service oriented architectures, component-based architectures, mashups and web applications.

Hye-young Paik is a senior lecturer at the School of Computer Science and Engineering in University of New South Wales (UNSW), Sydney, Australia. She received her PhD in Computer Science from UNSW in 2004. She is currently a senior member of Service Oriented Computing group at the school. Her research focus is in service oriented architectures and business process modeling and management.

Chae-Kyu Kim received BS in Mathematics from Korea University, then MS followed by PhD in computer science from University of Technology, Sydney (UTS) and University of Wollongong (UoW) Australia in 1993 and 1997, respectively. He has over 30 years of experience in research and development in various areas of computer systems and information management, and has been granted numerous national awards for his contributions in the industry. His research interests include media services, home networks, middleware and sensor networks. He is currently leading many research programs in IT Convergence Technology Research Laboratory at ETRI, Korea