2011 IFIP Ninth International Conference on Embedded and Ubiquitous Computing

RestThing: A Restful Web Service Infrastructure for Mash-up Physical and Web Resources

Weijun Qin, Qiang Li, Limin Sun, Hongsong Zhu
Institute of Software
Chinese Academy of Sciences
Beijing, China
qinweijun, liqiang, sunlimin, zhuhongsong@is.iscas.ac.cn

Yan Liu
School of Software and Microelectronics
Peking University
Beijing, China
ly@ss.pku.edu.cn

***Abstract*—In the field of Cyber Physical Systems and Pervasive Computing, physical resources and web resources can be easily handled and seamlessly integrated into our life. However, due to the heterogeneity of devices and tight coupling of individual information systems, the developers cannot easily create their specific applications by combining with physical and web resources. In this paper, we proposed RestThing which is a restful web service infrastructure based on REST principles in order to hide the heterogeneity of devices and provide a seamless way to integrate embedded devices with existing web applications. Besides, we implemented a prototyping system, which provided the restful accessible way of the wireless sensors, and built a demo application on the smart phone to collect and merge physical and web resources. Finally, we gave the performance evaluation of the prototyping system.**

***Keywords*-Cyber Physical System; mash-up; REST; RestThing; Web of Thing**

I. Introduction

The fields of Cyber Physical Systems [1], Pervasive and Ubiquitous Computing [2][3] to a large extent have been concerned so far with how to handle devices embedded into the physical world, and make them available to access by new applications. If there is a public information infrastructure where numerous embedded devices can be integrated and shared, people can design their own specific applications as the peer-produced contributors. For example, take Wikipedia. This kind of infrastructure will generate something of far greater value than the single individual system, where each contributor can build applications and exploit services to combine physical and web resources.

However, building such public information infrastructure is confronted with these problems. On the one hand, in order to meet the need of specific field, the existing information systems are independently designed and have unique characteristics. These systems adopted the specialized interfaces to make them available to access, which resulted in the effect of the 'isolated islands'. Since creating applications by using services provided by different information systems, developers should have professional knowledge and skilles in various technologies. On the other hand, unlike embedded devices comprising an application-specific system, components of the public infrastructure might be highly heterogeneous. There are various types of embedded devices, such as the wireless sensors, RFID, actuators and so on. These devices make use of specific communication protocols to connect to the Internet and have different capacities including various physical resources, computing capacities, communication bandwidth and capabilities of processing data. This heterogeneity would prevent developers from easily creating new applications. Although there nowadays are already some existing solutions to hide the heterogeneity and integrate enterprise information systems, they utilized the Big Web Services or WS-* [6], which are a set of complex standards and bring about high requirements for developers. For example, SenseWeb [4] and Cooltown [5] are such typical cases. However, WS-* protocol stack is already large and usually lead to build tightly coupled distributed systems.

To address these problems, we proposed RestThing - a restful web service infrastructure. RestThing based on Representational State Transfer [7] (REST) enables the devices to access and share in the entire Internet. The REST architectural style allows for the best decoupling of specific application scenarios and the basic handling of the resources. The embedded devices and Web information are both regarded as resources and manipulated by the uniform interface in REST architectural style. This infrastructure contains some key components: the restful API, embedded devices, web resources, the adaptation, service provider and applications. RestThing's goal is to enable developers to create many applications by complying with REST principles, which combine physical and web resources.

The contribution of this paper is given as follows. We proposed RestThing based on the REST paradigm. The infrastructure hides the heterogeneity of device and provides an easy and seamless way in which integrating embedded devices with existing applications. We have implemented a prototype system to exemplify and illustrate the advantages associated with the RestThing infrastructure. The prototype system contains wireless sensor network, the restful gateway and the mash-up application running on the smart phone.

The application combines the sensors and web resources (SINA micro-blog). Since using the uniform interface in REST principles inevitably would sacrifice some efficiency, we have evaluated the performance of this prototype system compared with these without using RestThing.

The rest of the paper is organized as follows: In section 2, we present a survey of related works. Section 3 introduces the overview of the restful web service infrastructure and proposes its key components to resolve the problems confronted. The design detail of the prototype system is presented in section 4. And in section 5, we have present evaluation performance of this prototype system. The future work and conclusion of the paper are presented at last.

II. Related Works

Coupling physical objects with the information applications is not a new topic, the work [8] introduced devices that perform some specific functionality and merged with computing capacity. Further, several attempts have explored web services to access and share the real-environment objects [9][10]. However, unlike the public information infrastructure, they provided special interface to operate these embedded devices.

SenseWeb [4], as the web infrastructure, used web services to share sensors in the entire Internet and hide heterogeneity of sensor network connectivity. It utilized the sensor gateway using Big Web Services [6] to connect sensor network to the Internet. Unfortunately, Big Web Services created tight coupled systems which are not sufficiently scalable, since it is based on a centralized repository. And Big Web Services or WS-* [6] are originally designed for integrating middleware and distributed systems by mapping their APIs to the Web information system. However, these standards used the Web as a transport infrastructure and acted as the extension of the basic API-oriented SOAP/WSDL model [11][20]. Therefore, along with the increasing functionality of middleware and distributed systems, the number standards and specifications of Big Web Services would become more complicated and larger. Obviously, the application developers cannot grasp these technologies in the short time. What they need is just low-enter barrier to quickly react to information or market requirements. Big Web Services cannot satisfy these requirements. Pachube [10] offers similar services as SenseWeb. However, they also are based on Big Web service and have adopted centralized repository, which brought about the high enter-barrier for application developers to enter into the business and caused the system scalability problem. These approach (e.g., [12], [13], [14]) implementing WS-* standards on devices are not appropriate for building the public infrastructure Our work differs from all these approaches above described. It is based on the Representational State Transfer (REST) paradigm, which is introduced by Roy Fielding's PHD thesis [7]. REST is capable of supporting to create a large number of applications, offering them simplicity and flexibility interaction. The uniform operation on resources is the principle of REST, which would lower the enter-barrier for developers. Compared with the work using the Big Web Services, our work used the HTTP as an application protocol, which provides a loose coupling style and a low barrier-to-entry to create new application for developers.

The first time to point out reusing well-accepted and understood REST principles was Erik Wilde's work [15], which proposed a path towards the Web where physical objects are made available through Restful principles. Reusing popular and successful Web style interconnected the embedded devices to the Web. TinyRest [16] also proposed to use the web to share the wireless sensors. It bridged the Web with the physical world through a restful gateway. However, TinyRest violated REST principles by introducing the extra verb 'subscribe'. Our prototype system also has utilized the restful gateway to connect sensors to the applications and complied REST principles. The recently work [17], [18] have proposed some prototypes of the Web of Things. These works have implemented the prototype and provided Restful API to explore functions in monitoring smart home information through a restful gateway. In these papers, sensors are capable of monitoring and controlling the energy consumption of household appliances that offer a Restful API to their functionality.

Our paper differs from these works in that we implement the prototype system that expanded a mash-up application running on the smart phone, which couples sensor nodes with web resources of micro-blog. Our work proposed RestThing infrastructure in which numerous developers can autonomously build their dedicated applications. We have expanded the application that can assemble physical and web resources by using restful API.

III. A Restful Service Framework

We have proposed RestThing -a restful web service infrastructure in this section (as seen Figure 1). RestThing's goal is that many applications can co-exist and access to embedded devices available just like web information resources. It adopted REST-driven and Resource-Oriented Architecture (ROA). In this infrastructure, the embedded devices and Web information are both regarded as resources and manipulated by the uniform interface. The infrastructure using ROA inherits all the outstanding mechanisms that made the Web scalable and successful like caching, load-balancing, indexing and searching and the stateless nature of the HTTP protocol. As any information exchange based on RestThing takes place through one of the four basic HTTP verbs (GET, POST, PUT and DELETE), the entire interaction with embedded devices can happen via browsers and browser side-kicks thereby virtually eliminating compatibility issues.

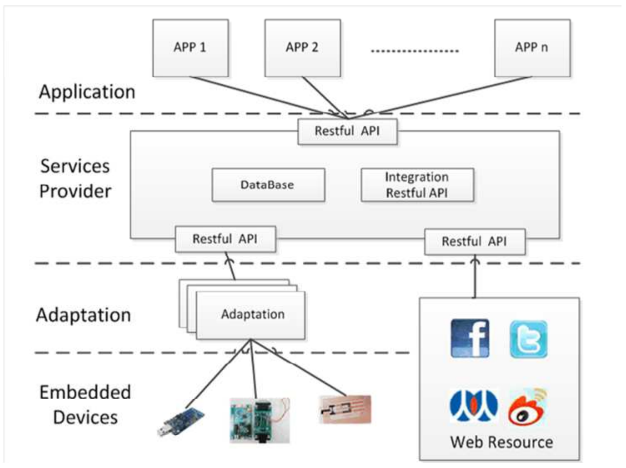


Figure 1. The restful service infrastructure.

The RestThing infrastructure's key components are Restful API, adaptation layer, embedded devices, web resources, the services provider and applications (see Figure 1).

A. Restful API

Restful API is the key factor in RestThing infrastructure to hide the heterogeneity of devices and provide accessible interface via the Web. Compared with Big Web Services, the Restful API based on REST architectural style allows for the best decoupling of specific application scenarios and the basic handling of the resources. The main idea of the RestThing infrastructure is to adopt a set of URI-addressable resources to define embedded devices and Web resources, and apply the uniform interface to design interaction functionality.

An important successful example using Restful API is the World Wide Web (WWW). The Web system identifies resources using Uniform Resource Identifier (URI) in a variety of identification schemes. The Hypertext Transfer Protocol (HTTP) [19] is the access method for interacting with resources in the WWW. HTTP is a rather simple protocol, and the design idea underlying this protocol has been described as REST style and has become an important foundation for many Web applications. HTTP acts as the main protocol for interacting with resources in a lightweight and loosely coupled way, and the Hypertext Markup Language (HTML) acts as the primary resource representation which is a lightweight and loosely coupled language providing hypermedia features. Therefore, the application fully blends into the Web, which simply provides access to resources through HTTP.

Considering the successful case of WWW, RestThing infrastructure applies HTTP as an application protocol to directly expose the functionalities of embedded devices by adopting REST-driven. Building application based on the RestThing infrastructure would lower the barrier of creating applications and eliminate compatibility issues. Applications access to various embedded devices just like Web resources by using uniform interface rather than various vendor-specific APIs.

B. Embedded Devices and Adaptation

There is great heterogeneity in the RestThing infrastructure, including various types of embedded devices and different capacities (such as wireless sensors, RFID, actuators, mobile phones). These devices have their dedicated communication protocols to connect to the Internet and are limited in computing capacity, communication bandwidth and capability of processing data.

Most embedded devices that nowadays collect physical information do not offer TCP/IP networking by default. For instance, wireless sensors use the IEEE 802.15.4 standards as a radio for communication. The visit to these devices demands an adaptation to support data transmission from the Web to the physical environment and vice-versa. Furthermore, there are other embedded devices that consist of a device driver module, which enables to support HTTP protocol. This driver module exposes embedded devices functionality via the Web and thereby establishes the link between the Web application and the device. For instance, sensor platforms based on Sun Small Programmable Object Technology (SPOT) are in favor of embedded HTTP server and directly connect the Internet via IEEE 802.15.4 link.

To hide much of these complexity and heterogeneity, embedded devices connect to the adaptation that provides Restful API to all components above it in our infrastructure. The adaptation provides connection and translation between service provider and physical resources. It implements specific methods to communicate with the embedded devices, and obtains data streams, submits data collection demands, or accesses sensor characteristics via Restful API. If the embedded devices do not support HTTP, the adaptation acts as the role of the gateway. The gateway transforms the special communication protocol to general HTTP. It receives control messages from service providers and translates them to the command understood by embedded devices. Similarly, it also collects data from devices and forwards them to service providers. If the embedded devices like Sun SPOT support HTTP link, the adaptation acts as the role of the driver module. It bridges the gap between the device and the Web, and directly exposes the functionality via the Web.

Considering devices mobility and environment uncertainty, the adaptation needs the devices management mechanism. The adaptation also needs to maintain some raw data in its local database for local applications.

C. Service Provider

The service provider is the central point of access in the RestThing infrastructure for sharing accessible embedded devices and creating applications. The functions of the service provider are internally divided between two components: the resource database and integration Restful API (see in Figure 1).

The resource database contains two kinds of information: the data collected from embedded devices and the data from existing Web resource (such as twitter, Facebook). As previously described, the adaptation transmits the data from embedded devices to the service provider in Restful API. The database is responsible for storing them and distinguishing their characteristics with other resources. The service provider offer data index and cache mechanism, which enable applications to quickly discover what resources are available. Considering the other kind information, existing web resources already enable to support normal operations in Restful API. These resources usually provide their own API for the third party developers to design applications, such as twitter's APIs. Therefore, the resource database only manages these available Restful APIs of Web resources and no need to store and index them.

The component of integration Restful API is responsible for generate more functional services than just adopting embedded devices or Web resources. Embedded devices and web resources are viewed the uniform resource form. Though there are great distinctions between embedded devices and existing Web resources, Restful API would hide this heterogeneity via the uniform operations in these resources. Therefore, there is no difference in accessing these resources by using Restful APIs. The component of integration Restful API organizes all the APIs no matter whether they come from embedded devices or web resources. It explores these APIs to developers for create more complicated functional services. When applications make use of both the embedded devices and web resources, these will be more attractive applications created by developers. For instance, if the service provider offer application peer-contributors the APIs about Facebook and the Restful API about the electricity data collected by embedded devices, a new created application will be created to share the electric power using condition of home equipment in Facebook.

IV. Prototyping Implementation

Here we describe how we implemented our prototyping system (in Figure 2) to exemplify and illustrate the advantages associated with the RestThing infrastructure (see Figure 1). We chose wireless sensor network as the representative of the embedded devices. A restful gateway support Ethernet interface and Zigbee communication, which are lightweight, and scalability components. The service deployed in the personal computer, which provide the Restful API both sensor network and Web resources. In the prototype system, SINA micro-blog acts as the sample of Web resources, which is similar to Twitter and popular in China. The new application with combining physical and web resources runs on the smart phone, which has the operation system of Android 2.1. The overall architecture of the prototype system is described in Figure 2.

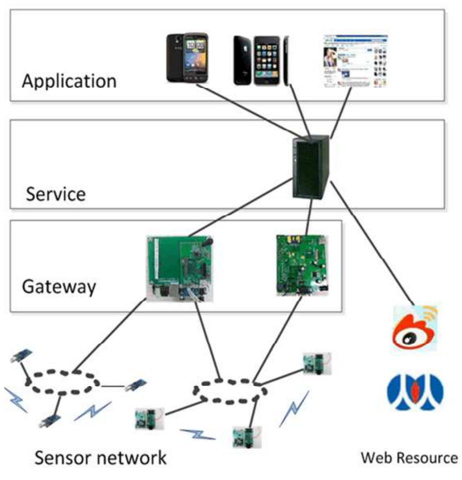


Figure 2. The prototyping system architecture.

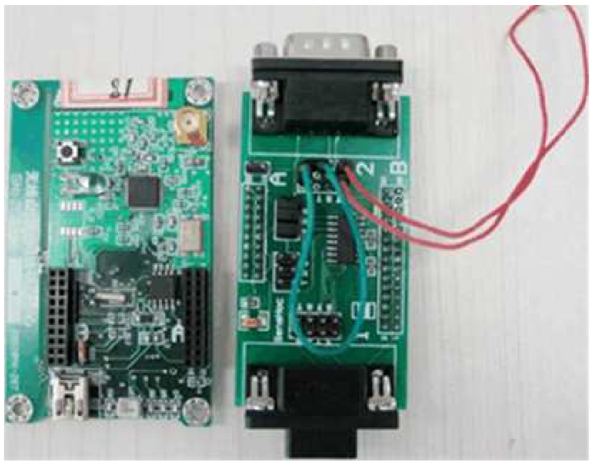


Figure 3. The sensor node of Telosb.

A. Wireless Sensor Network

We decided to select sensor motes to represent embedded devices in our implementation prototype system, since they are very easy to program and offer some basic sensing capabilities. These sensors are equipped with a 250kbps, 2.4GHz, IEEE802.15.4-compliant Chip CC2420 Radio. IEEE802.15.4 is an IEEE standard that defines a MAC and PHY layer targeted to Wireless Sensor Networks (WSN). The node of Telosb that we have used in the experiment is shown in Figure 3.

The sensors that measure various physical variables are foundational element of the prototype system. Since the new application uses the environment data observed by sensors and sensors not offer TCP/IP networking by default, we have deployed the restful gateway to handle Restful API on these devices. We abandon the idea of having an HTTP server

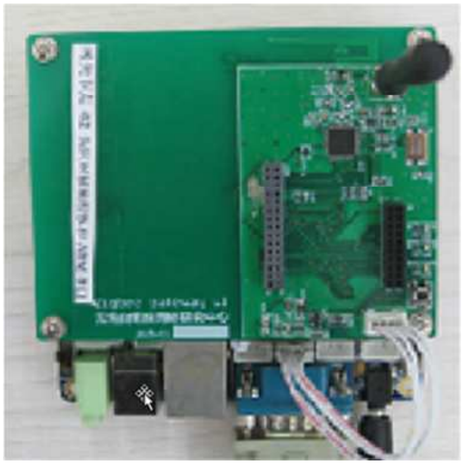


Figure 4. The restful gateway in the experiment.

directly on the device and the sensors are accessed through the restful gateway. Therefore, new application can easily be created by indirectly using sensors Restful API.

B. The Restful Gateway

Due to the fact that the mote sensor used in the prototype system do not offer TCP/IP networking by default but Zigbee, our prototype system makes use of the restful gateway handle HTTP requests. We have chosen the gateway based ARM9 32-bit RISC processor architecture, S3C2440 400MHz CPU processor and 64M Flash memory. To hide much of this complexity, the Restful gateway provides a uniform interface to all components above it and manages all sensors in its scope. Applications can access the gateway to obtain sensor data streams, submit data collection demands, or access sensor characteristics through the Restful API. The gateway that we have used in the experiment is shown in Figure 4.

The restful gateway implements the restful API using http protocol. And sensors and the web information are view as resources and manipulated by the uniform Restful APIs in the HTTP standard. Accessing the sensors used the four basic HTTP verbs (GET, POST, PUT and DELETE). The detail descriptions amongst these four verbs are following:

- GET: It is used to retrieve a representation of a resource.Many Web applications support only this verb which returns a representation of the requested resource. For example, sending a GET request along with the URI http://.../sensors/sensorid/ dataType. mediaType would return the data observed by the sensor.
- POST: It is a method to alter the state of a resource. By using POST, it is possible to update a resource with new information. An update can either change the state of an existing resource, or it can cause the creation of a new resource. These two cases can be distinguished by the status code returned in the HTTP response.

Table I
A LIST OF THE RESTFUL APIS

Resource	REST Verb	MIME Type	URI
SenCurrentRes	GET	XML/JSON	sensors/{sensorid}/ {dataType}.{mediaType}
SenHistoryRes	GET	XML/JSON	sensors/history/{sensorid}/ {dataType}/{fromTime}/ {toTime}.{mediaType}
SenActiveRes	GET	XML/JSON	sensors/activeNode/ all.{mediaType}
SenCmdRes	POST	XML/JSON	sensors/{sensorid}/ command/{dataType}/interval

- PUT: This verb creates a new resource with a name that is specified by the client. This requires that the naming scheme of resources is well-known by the client, the client has sufficient access rights, and the client supplies a representation of the new resource which is sufficient to instantiate that resource.
- DELETE: If a resource is no longer required, this method removes the URI from the accessible resources of a server. This does not necessarily mean that the resource itself will be deleted and simply means that the URI that has identified the resource will no longer work.

In Table I, there is the specific description of HTTP verbs about operation on the mote sensors in our prototype system. We define four types of URI as the identification of sensors: SenCurrentRes is the root identification of current resources; SenHistoryRes is the identification that predicates data stored in the cache or database of the gateway; SenActiveRes is the identification that indicates the activity sensor nodes; SenCmdRes is the identification that indicates commands altering the state of physical devices. The first three resources have the same HTTP verbs, which GET is used to retrieve a representation of a resource. The fourth actuator has POST verbs, which is a method to alter the state of a resource. All the resource URIs has two types presentations. XML [22] is the default standard for structured information on the Web. Application scenarios can define their own schemas for XML, and in many cases standards or standards exist and can be reused. The JavaScript Object Notation (JSON) [23] provides a better solution for JavaScript environments. This representation may be more limited than XML, but can be a good way to make resource data available for Web 2.0 applications.

The restful gateway receives a Restful HTTP verb and URL from the Web application and interprets it to the format understood by the sensor. The gateway encapsulates data observed by the sensor to the presentation as XML or JSON and provides them to the Web application.

C. Application Demonstration

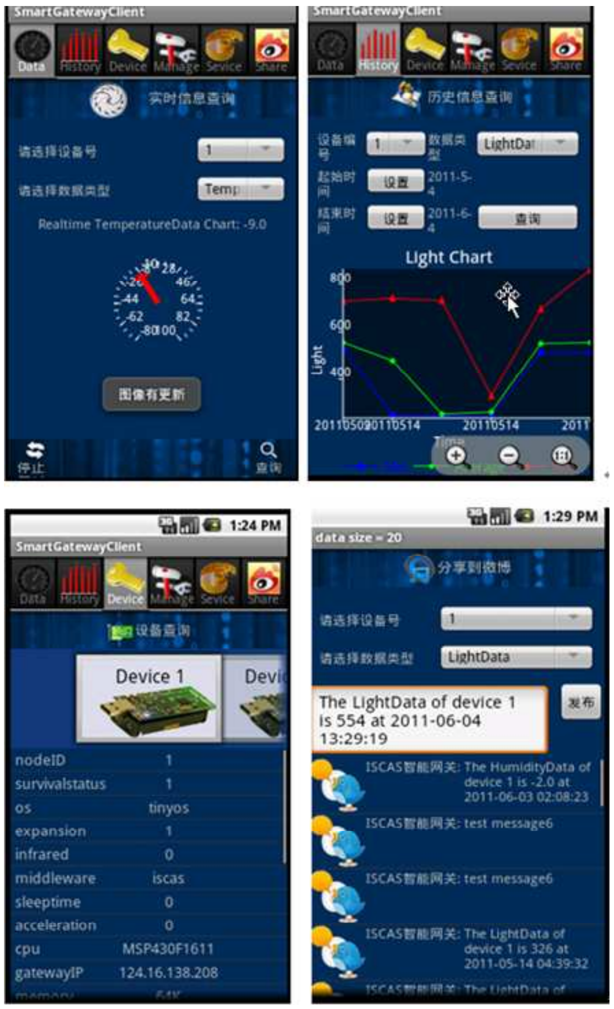
In this section, we have established application demonstration using services which are provided by the restful gateway. The application runs a mobile phone (HTC Wildfire G8), using android 2.1 operation system. Android is an open source mobile operating system initiated by Google, which is built on a customized Linux kernel, libraries and an extensible Application Framework developed in the Java language but running on a special dalvik virtual machine. Restlet [21] has been transplanted on Android with both the client-side and the server-side HTTP connectors.

We have created the mobile phone application using the services provided by the restful gateway. The application combines physical and Web resources in the Restful API. The user interface of the application running the smart phone is shown as Figure 5, including four views: the real-time data view, the historical data view, the device management view and the share services view.

The real-time data view is used to obtain current data from the wireless sensor network. The smart phone updates the current environment data every 10 seconds through sending GET URL requirement to the restful gateway. The gateway will give the response as the feedback to the phone. As shown in Figure 5, we have chosen the temperature sensor whose device number is one to get current laboratory indoor environmental temperature. The real-time data view displays the current temperature information in the form of the compass.

The historical data view is used to present the historical environment information. The restful gateway collected the data observed by the sensors and stored them in the database. We have decided the phone to support check the scope of time from one hour to one month, since too long or too short time is no significant for observation. When phone user wants to check the history information, the smart phone sends the GET URL requirement to the restful gateway. As shown in Figure 5, we have chosen the light information in the laboratory indoor environment from May 4 to June 4. The gateway queried data in this segment time and gave the response to the smart phone. The historical data view presents the historical information through drawing a curve graph in the fixed coordinate.

The device management view provides users with an overview of current activity sensor nodes. The restful gateway checks the state condition of sensor nodes in its managed area. If there is one sensor node breakdown with energy depletion, the gateway will record this information and send it to the smart phone. The user can send GET URL to get information or PUT URL to change the status of the device. As illustrated in Figure 5, we have chosen to check the status of the device number one. The restful gateway received the check message response from this device and sent it to the smart phone. The device management view offers



nodeID	1
survivalstatus	1
os	tinyos
expansion	1
infrared	0
middleware	iscas
sleeptime	0
acceleration	0
cpu	MSP430F1611
gatewayIP	124.16.138.208
memory	64K

The LightData of device 1 is 554 at 2011-06-04 13:29:19

ISCAS智能网关: The HumidityData of device 1 is -2.0 at 2011-06-03 02:08:23

ISCAS智能网关: test message6

ISCAS智能网关: test message6

ISCAS智能网关: The LightData of device 1 is 326 at 2011-05-14 04:39:32

ISCAS智能网关: The LightData of

Figure 5. The top UI (left to right): Real-time data view and Historical data view; the bottom UI (left to right): Device management view, and Share services view.

the information of the selected device and the available operation to the devices.

The share services view presents the services combining physical and Web resources. The restful gateway provided the URL interface to operate sensors. We have use SINA micro-blog as Web resources, which is similar to Twitter. SINA micro-blog provides the operation APIs for the third party to develop applications. Additionally, Sina micro-blog is very popular in China. The smart phone application makes use of these two kinds resource: physical environment and micro-blog information. The smart phone user can share his around environment information observed by embedded devices in the micro-blog. As illustrated in Figure 5, we have chosen to publish my room light information in my micro-blog, and my micro-blog followers can at once know this news.

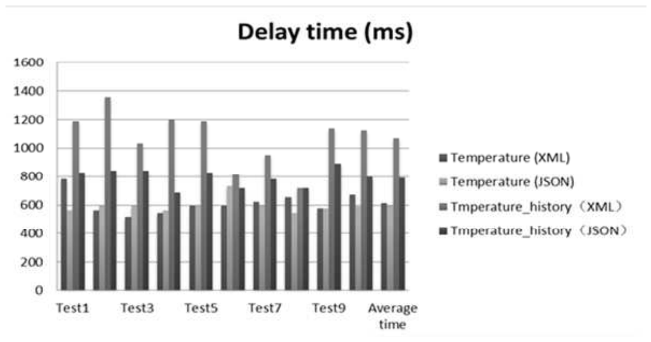


Figure 6. Time is the duration from the creation of a request in the client to the arrival of the response back by smart gateway. Ten times tests and average time are shown in it.

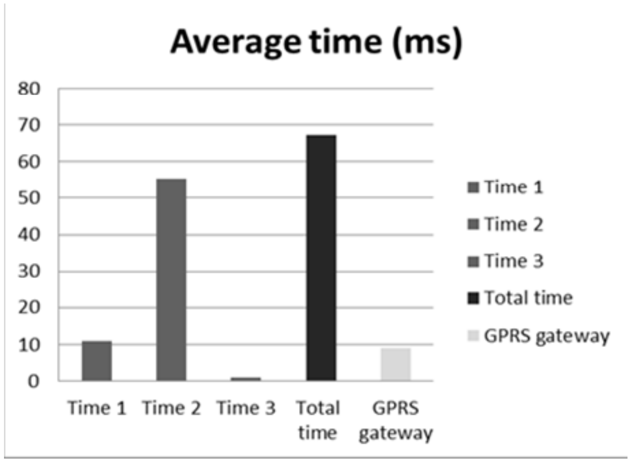


Figure 7. Testing at each stage consuming time of the smart gateway and the normal gateway.

V. Performance Evaluation

Using the uniform interface would inevitably sacrifice efficiency specially compared with these gateway just transmit sensor data without using URL. Therefore, we have evaluated the performance of the prototype system through testing the amount of delay time from a request to the back response. As a compared target, we have also gain another delay time which through a normal gateway just translating sensor data to the Internet.

We have set the restful gateway and the service provider into the same personal computer. Around it, we deployed fixed scale of ten sensors. The sink node was plugged in one of the USB ports of the personal computer to serve as a base station, to forward IEEE802.15.4 wireless packets from/to the personal computer through the serial-over-USB port. Since the internet delay is dynamic, the client and the smart gateway using the same local IP address. We focus on gain the delay time from the smart gateway serial port reading date to the client getting the response. We performed 10 times test and gained their average time in the experiment.

Figure 6 presents the results of the experiment executions delay time. There are four test experiments: temperature real-time date with XML presentation, temperature real-time date with JSON presentation, temperature history date with XML presentation, temperature history date with JSON presentation. Since the temperature history back response has more than 1000 items of data compared with the current data back response, it has longer delay time in the experiment. When the amount of data is small (real-time data response is small amount), JSON and XML presentations have more or less time delay. If the data amount is large, using JSON as resource presentation has less time delay than XML.

Further, we have divided the delay time consumed in the restful gateway into three stages, which includes the time reading from the serial-over-USB port and transformation data from Zigbee format to Ethernet format, the time stored in database, the time encapsulated data to the URL's presentation. As a compared target, the normal gateway didn't comply with REST principles, which just contains read date from serial port and data transformation. Ttotal is the total time consuming in the restful gateway, T1 is the time that read data by the serial port and data interpret; T2 is the time storing the data into database or cache, T3 is the time that encapsulated data to resource presentation. We performed ten times test and gain their average time in the experiment. Figure 7 presents the results of the experiment executions. Since the restful gateway have additional database operation, which cost most time, it had longer delay time cost than the normal gateway. After all, the performance of the gateway is acceptable.

VI. Conclusion and Future Works

In this paper, we have proposed RestThing to access the device data streams and shared across the entire Internet. The RestThing infrastructure is based on REST paradigm, which offers a uniform, efficient and standardized way of interacting with information applications. It hides the heterogeneity of devices and integrated isolated information systems. Developers can easily build their dedicated applications. To illustrate the advantages of the RestThing, the prototype system has been implemented, which contain wireless sensor network, the restful gateway and the mashup application running on the smart phone. However, using the uniform interface would inevitably sacrifice efficiency specially compared with these gateway just transmit sensor data without using URL. We have evaluated the performance of the prototype system.

There are some future works we should do. Firstly, the gateway runs in the ad-hoc environment or sharing public infrastructure, where the embedded devices have limited resource and constraint computing capacity. The RestThing is required to present the effective management mechanism. Secondly, the URLs not directly connect to the embedded devices, and the adaptation act as the role of the transit station. The synchronization mechanism should be developed to guarantee URL link and data transmission real-time. At last, the RestThing infrastructure cannot obtain a composite service which can provide additional functions, or remove part of subservices to get a simplified service which is convenient to use. We consider adding standardized service description in the service infrastructure in the future work.

Acknowledgment

This work has been funded by State Key Development Program for Basic Research of China No.2011CB302902, National Major Special Science and Technology Projects No.2011ZX03005-006, and National Natural Science Foundation of China No.61073014.

References

[1] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, ”Cyber-physical systems: the next computing revolution,” in *Proceedings of the 47th Design Automation Conference*, Anaheim, California, 2010, pp. 731-736.

[2] M. Weiser, ”Some Computer Science Issues in Ubiquitous Computing,” *Communications of the ACM*, vol. 36, no. 7, pp. 75-84, 1993.

[3] M. Armbrust, A. Fox, R. Griffith, et al., ”Above the Clouds: A Berkeley View of Cloud Computing,” Tech. Rep. UCB/EECS-2009-28, 2009.

[4] A. Kansal, S. Nath, J. Liu, and F. Zhao, ”Senseweb: an infrastructure for shared sensing,” *IEEE Multimedia*, vol. 14, no. 4, pp. 8-13, 2007.

[5] T. Kindberg, J. Barton, G. Becker, D. Caswell, and P. Debaty, ”People, places, things: web presence for the real world,” *Mob. Netw. Appl.*, vol. 7, no. 5, pp. 365-376, 2002.

[6] C. Pautasso, O. Zimmermann, and F. Leymann, ”RESTful Web Services vs. ”big” web services: making the Right Architectural Decision,” in *Proceedings of the 17th International World Wide Web Conference*, Beijing, China, 2008, pp. 805-814.

[7] R.T. Fielding, ”Architectural Styles and the Design of Network-based Software architectures,” PhD.thesis, University of California, Irvine, Irvine, California, USA, 2000.

[8] E. Bergman, *Information Appliances and Beyond*, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2000.

[9] J.I. Vazquez, A. Almeida, I. Doamo, et al., ”Flexeo: An Architecture for Integrating Wireless Sensor Networks into the Internet of Things,” *Advances in Intelligent and Soft Computing*, vol. 51, no. 2009, 2009, pp. 219-228.

[10] V. Trifa, S. Wieland, D. Guinard, and T.M. 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, June 2009.

[11] WSDL, ”Web services description language,” 2001. http://www.w3.org/TR/wsdl.

[12] N.B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, ”Tiny web services: design and implementation of interoperable and evolvable sensor networks,” In *SenSys 2008*, 2008, pp. 253-266.

[13] F. Jammes, and H. Smit, ”Service-Oriented Paradigms in Industrial Automation,” *IEEE Trans. Industrial Informatics*, vol. 1, no. 1, pp. 62-70, Feb. 2005.

[14] L.M.S. de Souza, P. Spiess, D. Guinard, M. Köhler, S. Karnouskos, and D. Savio, ”SOCRADES: A Web Service Based Shop Floor Integration Infrastructure,” in *Proc. Internet of Things Conf. (IoT 2008)*, 2008, pp. 50-67.

[15] E. Wilde, ”Putting Things to REST,” Technical Report UCB iSchool Report 2007-015, School of Information, UC Berkeley, November 2007.

[16] T. Luckenbach, P. Gober, S. Arbanowski, A. Kotsopoulos, and K. Kim, ”TinyREST -a protocol for integrating sensor networks into the internet,” in *Proc. of REALWSN*, 2005.

[17] D. Guinard, and V. Trifa, ”Towards the web of things: Web mashups for embedded devices,” in *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), Proceedings of WWW (International World Wide Web Conferences)*, Madrid, Spain, April 2009.

[18] M. Müller, ”RESTful EPCIS - Design and Implementation of a Web-enabled Electronic Product Code Information Service (EPCIS),” M. thesis, University of Fribourg, 2009.

[19] R.T. Fielding, J. Gettys, J.C. Mogul, H.F. Nielsen, L. Masinter, P.J. Leach, and T. Berners-Lee, ”Hypertext Transfer Protocol–HTTP/1.1.” Internet proposed standard RFC 2616, June 1999.

[20] N. Mitra, and Y. Lafon, ”SOAP Version 1.2 Part 0: Primer (Second Edition).” World Wide Web Consortium, Recommendation REC-soap12-part0-20070427, April 2007.

[21] Restlet, ”Restlet, lightweight rest framework for java,” 2007. http://www.restlet.org/.

[22] S.J. DeRose, E. Maler, and D. Orchard, ”XML Linking Language (XLink) Version 1.0,” World Wide Web Consortium, Recommendation REC-xlink-20010627, June 2001.

[23] D. Crockford, ”The application/json Media Type for JavaScript Object Notation (JSON).” Internet informational RFC 4627, July 2006.