The Web of Things: A Survey
(Invited Paper)

Deze Zeng, Song Guo, and Zixue Cheng
School of Computer Science and Engineering, The University of Aizu, Japan
Email: {d8112106, sguo, z-cheng}@u-aizu.ac.jp

Abstract— **In the vision of the Internet of Things (IoT), an increasing number of embedded devices of all sorts (e.g., sensors, mobile phones, cameras, smart meters, smart cars, traffic lights, smart home appliances, etc.) are now capable of communicating and sharing data over the Internet. Although the concept of using embedded systems to control devices, tools and appliances has been proposed for almost decades now, with every new generation, the ever-increasing capabilities of computation and communication pose new opportunities, but also new challenges. As IoT becomes an active research area, different methods from various points of view have been explored to promote the development and popularity of IoT. One trend is viewing IoT as Web of Things (WoT) where the open Web standards are supported for information sharing and device interoperation. By penetrating smart things into existing Web, the conventional web services are enriched with physical world services. This WoT vision enables a new way of narrowing the barrier between virtual and physical worlds. In this paper, we elaborate the architecture and some key enabling technologies of WoT. Some pioneer open platforms and prototypes are also illustrated. The most recent research results are carefully summarized. Furthermore, many systematic comparisons are made to provide the insight in the evolution and future of WoT. Finally, we point out some open challenging issues that shall be faced and tackled by research community.**

Index Terms— **Internet of Things, Web of Things, Survey**

I. Introduction

Ubiquitous computing (a.k.a., pervasive computing), which has been extensively studied for many years, is experiencing radical changes recently as the physical world devices, e.g., home appliances and industrial machines, are becoming smart thanks to the progress in computing technology development. In parallel, the communication techniques also make much progress recently. Internet access will very likely become commonly accessible by those "smart things", motivating the concept of *Internet of Things (IoT)*.

IoT is regarded as the next big possibility and challenge to the Internet. The Internet will be no longer just a network of computers, but will potentially involve trillions of smart things with embedded systems. IoT will greatly increase the size and scope of current Internet, providing new design opportunities and challenges. Internet with smart things is generally viewed as a constrained IP network with limited packet size, high degree of packet loss, and even intermittent connectivity and characterized by severe limits on throughput, available power, and particularly the complexity that can be supported. A variety of recent research activities have been launched to address those challenging issues, from the technological to the social aspects. In particular, a central issue focuses on how to make a full interoperability of interconnected devices possible, to provide them with an always higher degree of smartness by enabling their adaptation and autonomous behavior while guaranteeing trust, privacy, and security [1].

Currently, the web has already become the major medium of communication in today's Internet. On the other hand, tiny web server technology has been researched for decades and now various embedded tiny web servers are available. More specifically, web services have been proven to be indispensable in creating interoperable applications on today's Internet. Smart things with embedded web servers can be abstracted as web services and seamlessly integrated into the existing web. It is natural to reuse existing web technologies and standards to unify the cyber-world and the physical-world. As a result, one research trend treats IoT as *Web of Things (WoT)*. As existing web technologies can be reused and adapted to build new applications and services with participation of smart things. This yields higher flexibility, customization and productivity. In brief, different from traditional view of IoT which gives everyday device an IP address and makes them interconnected on the Internet, WoT enables them to speak the same language, so as to communicate and interoperate freely on the Web.

The WoT vision depicts a view where a collection of web services that could be discovered, composed and executed. Thus enriches the scope of traditional web services by promoting the web from only cyber-world services to both cyber-world and physical-world services. Furthermore, WoT actually is an ecosystem of services not only about adding more services in but more about orchestrating various kinds of services in a graceful manner, making the services more human-centric and intelligent.

Let us use an example to illustrate the concept of WoT. After the nuclear leakage accident happened at Fukushima Dai-ichi Nuclear Power Plant on 11 March, 2011 after the devastative tsunami, people have concerned the radiation level at each places. A Japan Geigermap has then been developed by integrating Google Map web service and geiger counter readings. It provides a new web service which visualizes the crowd-sourced radiation geiger counter readings across Japan on a Google map. A snapshot is taken as shown in Fig. 1[1].

Manuscript received February 15, 2011; revised May 15, 2011; accepted June 15, 2011.

[1] http://japan.failedrobot.com/

doi:10.4304/jcm.6.6.424-438

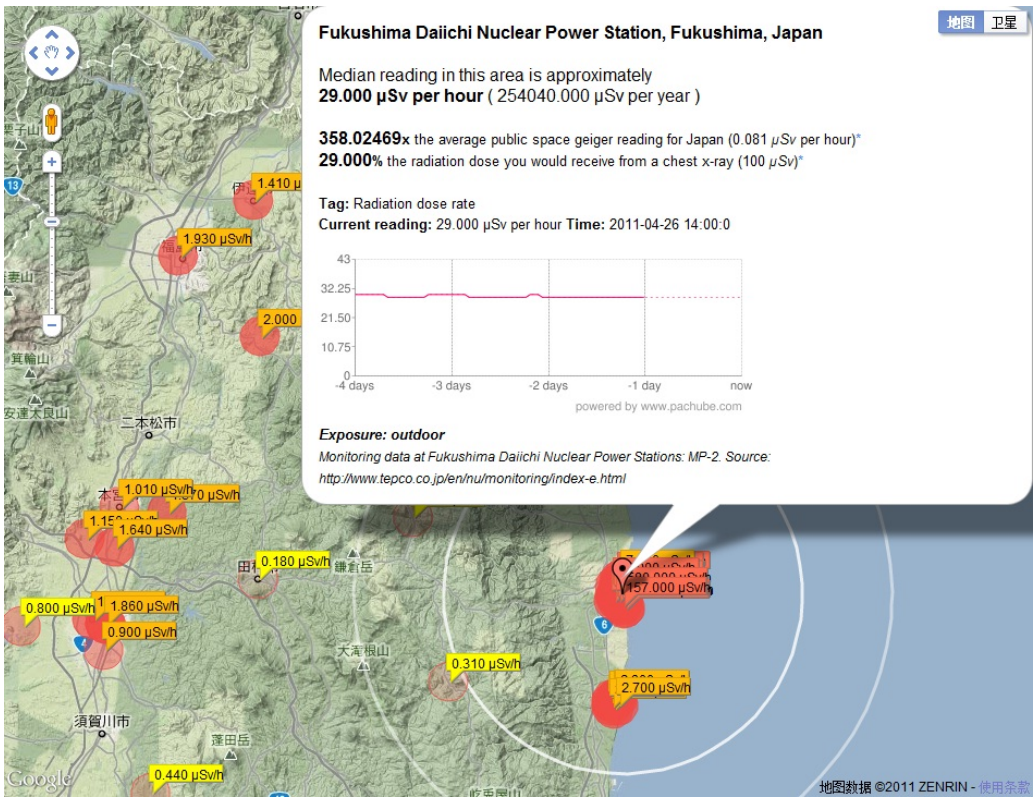

Figure 1. Snapshot of Geigermap around Fukushima Daiichi Nuclear Power Station, Fukushima, Japan

The Japan Geigermap is only a simple and straightforward example of WoT about data sharing. The power of WoT is far more beyond data sharing. The development of WoT is still at an initial stage, and there are still many issues to be tackled to fully exploit the potential of WoT. The inclusion of services from smart things makes WoT different from the traditional web. For example, the embedded tiny web servers are not as powerful as traditional ones. The service may not be always available as traction-al one due to the intermittent connectivity caused by duty cycle. Furthermore, traditional web protocols might not be suitable to provide services in the low-power and lossy networks consisting of resource constrained smart things. All these problems pose new research challenges and call for new efficient solutions. Many research activities have been actively conducted toward the solutions that fulfill such highlighted technological requirements. The main object of this survey is to give readers an overview of WoT, the state of WoT development, the potential of WoT, and the key issues that remain to be tackled. The remainder of the paper is organized as follows. In Section II, we briefly give a glance at the motivation as well as the basic concept of WoT. In Section III, we introduce the architecture about WoT including the integration methods and the web service paradigms. In Section IV, the main enabling technologies to WoT are presented. In Section V, some existing open WoT platforms and WoT application prototypes are introduced. Section VII concludes this survey work.

II. Overview of the Web of Things

The communication for smart things has been studied for decades. Several different technologies and standards have been proposed in this area. Making the smart things interconnectable such that bits can be transferred between devices is only the first step, more works are expected to make smart things interoperable such that they are understandable with each other. Interoperability is particularly essential, and a must, to build system with various devices, especially those from different manufacturers. Let us first review some of those major technologies about the interoperability issue.

Universal Plug and Play (UPnP) is a suite of networking protocols extended from the idea of the original Plug and Play to a networked system context. It was promoted by the UPnP forum[2] mainly for personal networks devices to discover each other's presence and further to establish connections on the network. UPnP is based on established protocols and standards, such as TCP/IP, UDP, HTTP, HTTPU (HTTP over UDP), SOAP, WSDL, etc. Currently, UPnP is the most popular solution for personal network implementation. However, UPnP has several drawbacks [2]:

1) There is no authentication protocol proposed for UPnP. Any devices are allowed to configure the other devices of the personal network, without any user control, resulting in a critical security issue when the smart things are available on the Internet.

[2]http://www.upnp.org

2) UPnP is not strictly standardized as some UPnP devices are based unstandardized protocols such as HTTPU, restricting its universal interconnection somehow.
3) UPnP is inapplicable to some resource-constrained devices because it normally uses a lot of heavy protocols (e.g., SOAP, WSDL, etc.) involving complex processing.

Alternatively, the JXTA technology[3] is proposed as a solution for peer-to-peer applications design, enabling interconnections of heterogeneous devices into a same network. Later on, a C language based version, JXTA-C was proposed in order to embed JXTA into resource-constrained devices [3]. Unfortunately, JXTA protocols have not been standardized and have not been widely accepted for embedded devices in industry either.

One trend is integrating the devices into the Web. It has been found that the web severs can be built in a size of only a few KBs [2], [4], [5]. It is possible to integrate the web servers into many devices directly. Those devices then proactively serve their functionality over the Web. Using the free, open, flexible, and scalable Web as the universal platform to integrate smart devices outperforms all other solutions mentioned earlier in terms of easiness, flexibility, customization and security. This idea has attracted much attention from both academia and industry, especially after the IoT concept emerges recently.

The web browsers have been available on almost any platform, from computers to PDAs, smart phones, and tablets, and become the de facto standard user interface to a variety of applications. The Web-enabled applications can be accessed from any location provided there is an Internet connection. Applied to embedded systems, web technologies can offer platform-independent interfaces such that the end-users do not need to install specific softwares and drivers for different devices. Also, developers do not have to tediously develop different softwares and drivers targeting different platforms for one thing. The Web provides a one-for-all solution. An overview of of WoT vision is shown in Fig. 2.

Furthermore, although devices become programmable, providing great opportunities to create more innovative and powerful applications, development, especially composition, of applications that run on top of those physical devices is still a cumbersome process as it requires extensive expert knowledge (e.g. specific APIs in a specific programming language) about all different physical devices. This more or less constrains development of smart things based services. Fortunately, existing web technologies (e.g. mashup), which previously targeted for cyber-world web services can be reused for application development with the participation of physical smart things provided that they can be abstracted as web services. By reusing existing web technologies, the expenses for additional infrastructure and overall implementation time can be minimized. These technologies can promote the progress of IoT significantly.

[3]http://java.sun.com/othertech/jxta/

III. Web-oriented Architecture

A general architecture of WoT is illustrated in Fig. 2. Reusing existing web architecture as the basic platform, some smart things act as web servers and directly provide web services on the web. WoT has a flat architecture, compared to the traditional server-client architecture. Two issues need consideration in such an architecture: how to integrate the physical things to the web and how to make the physical things provide composable and interoperable web services.

A. Integrating Smart Things to the Web

As shown in Fig. 2, there are two optional methods to integrate things to the Web: direct integration and indirection indirection [6]. For example, the home appliances in the figure can be viewed as directly integration while the RFIDs are indirectly integrated through a RFID reader with an embedded server. Usually a system may not solely rely on a single method, but may use both methods as a hybrid way.

1) Direct integration: To directly integrate things to the Web, it is first required that all the things must be addressable, i.e. everything must have an IP address, or must be IP-enabled when connected to the Internet. WoT also requires connectivity and interoperability at the application layer. Web server shall be embedded such that things can understand each other through the web language specified by web standards. With the development in both communication and computation technologies, it is likely that more devices will become IP-enabled and can be embedded with web server. Those devices can be directly integrated into the Web and abstracted as web services. Thus, they can directly communicate with people from any terminal with a standard web browser. Other devices can also interoperate with them through standard web operations, e.g. GET and POST.

Many pioneer solutions have been provided to directly integrate smart things to the Web. Guinard et al. [6] present a prototype directly integrating IP-enabled Sun SPOT with web server. Each device in their prototype offers its functionality through a web API. Akribopoulos et al. [7] introduce an architecture where all the small programmable objects are integrated through web services where the Sun SPOT applications and sensor data are uniformly exposed through web services. They avoid employment of additional gateways by using TCP/IP protocol in the devices directly. Also a prototype is implemented using Sun SPOT. Ostermaier et al. [8] present a prototype using programmable low-power WiFi modules for connecting things directly to the web. They leverage the ubiquity of IEEE 802.11 access points and the interoperability of the HTTP protocol. Using a loosely coupled approach, they enable seamless association of sensors, actuators, and everyday objects with each other and with the Web. All those works demonstrate convincingly that

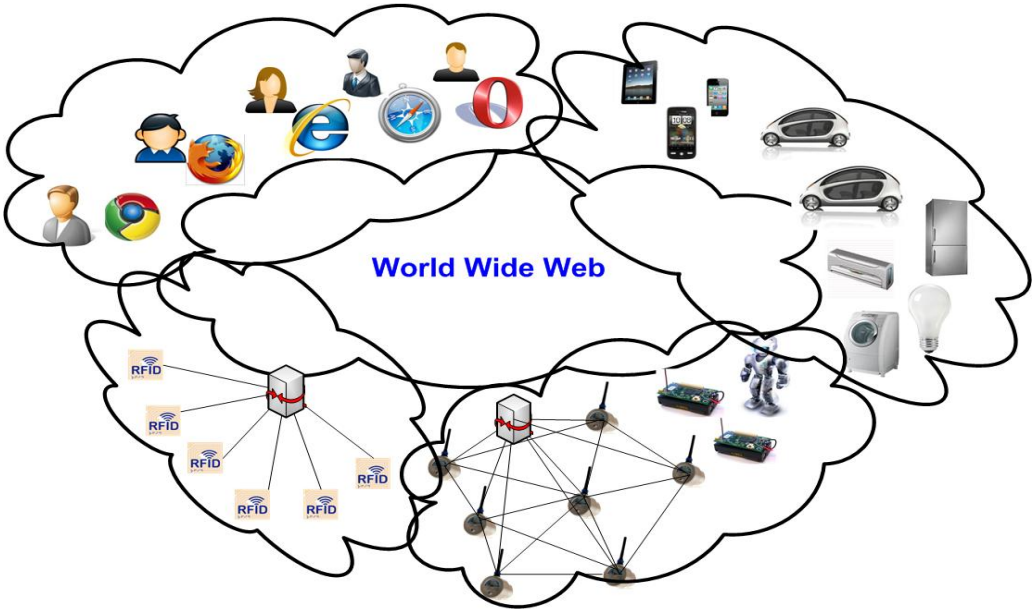

Figure 2. Overview of Web of Things

it is possible to integrate smart things directly into the Web now.

2) Indirect integration: However, not all devices can be powerful enough to be embedded with web server. Some devices are with too limited resource to allow web server embedded, such as RFID tags. On the other hand, sometimes there is no need to directly integrate all the smart things (e.g. sensor nodes in a sensor network) into the Web in the consideration of cost, energy and security. For both cases, a different pattern, indirect integration, can be adopted. In this pattern, an intermediate proxy locates between the smart things and the Web. The proxy is usually called smart gateway. To the smart things (inward), the smart gateway communicate with the smart things (e.g. reading from RFIDs) and therefore shall understand the proprietary protocols of the smart things; to the Web (outward), it abstracts the proprietary protocols or native APIs of smart things and offer uniform accessible web APIs over the Web.

Several prototypes with smart gateways to directly integrate smart things into the Web have been published in the literature. Hwang et al. [9] design a smart sensor gateway for sensing data aggregation and sensor network management. To enable using the web browser to efficiently query and manage the sensor network, the sensor gateway is embedded with a web server supporting HTTP1.1 protocol. The authors also implemented Java applet for dynamic and efficient data exchange. Trifa et al. [10] implement smart gateway for web-based interaction and management of embedded devices. The gateway enable accessing to sensor networks through a lightweight web service interface. In [11], the authors build an EPC Network prototype by using virtualization, cloud computing and web technologies. In their prototype, the RFID reader behaves like a smart gateway which locates between the cloud server and RFID tags.

B. Web service paradigms

As we are able to integrate different smart things with various capabilities intto the Web, the next logical step we shall consider is how to abstract those devices into reusable web services other than simple static or dynamic web pages. Web services are defined by the World Wide Web Consortium (W3C) as a software system designed to support interoperable machine-to-machine (M2M) communications over a network. As W3C states, there are two major paradigms of web services: REST-compliant Web services and arbitrary Web services [12]. The primary purpose of the service is to manipulate web resources using a uniform set of "stateless" operations in the former one while using an arbitrary set of operations in the latter one. Both paradigms can be adopted by smart things or smart gateways.

1) WS- Architecture:* It is usually referred as WS-* for Web Services that use Simple Object Access Protocol (SOAP) messages with an Extensible Markup Language (XML) payload and a HTTP-based transport protocol to provide remote procedure-calls (RPCs) between clients and servers. It has been popular in traditional enterprises and widely used in enterprise machine-to-machine (M2M) systems. The key technologies of WS-* are SOAP, Web Service Description Language (WSDL), Universal Description Discovery and Integration(UDDI) and Business Process Execution Language (BPEL).

SOAP [13] is an XML-based protocol to let applications exchange information over HTTP. A SOAP interface is typically designed with a single URL that implements several RPCs methods, which define a message architecture and format, hence providing a rudimentary processing protocol. The top-level XML element of SOAP message is called *envelop*, which includes two XML elements: *header* and *body*. The *header* specifies routing and Quality of Service (QoS) configuration while the *body* contains the

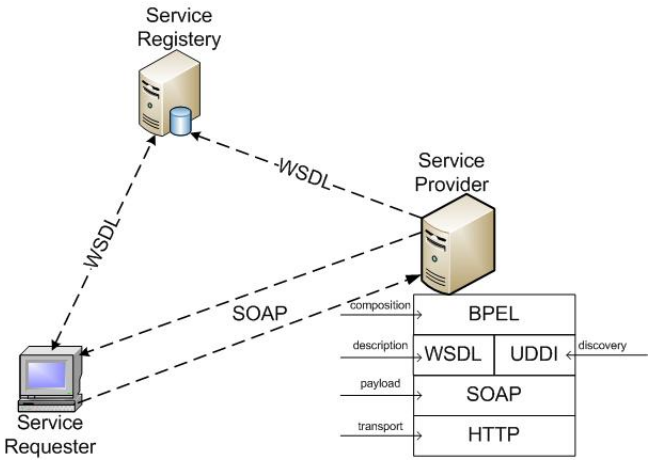

Figure 3. WS-* workflow and Protocol Stack

payload of the message indicating the interoperations.

WSDL [14] is an XML-based language describing Web services as a collection of communication end points that can exchange messages. In other words, a WSDL document describes a Web service's interface and provides users with a point of contact. The SOAP messages and sequences are abstractly described by WSDL. A WSDL *port type* contains an abstract set of operations supported by endpoints. The WSDL *binding* links the set of abstract operations with concrete protocol and data format specification for a particular port type. WSDL describes service interface, which are independent of the service implementation endpoint and how the services are implemented.

UDDI [15] is a platform-independent, XML-based registry framework for describing and discovering worldwide Web services. It can be viewed as a directory of WSDL-described web services. Web services can be registered and located in the directory. It can be requested using SOAP messages to provide access to WSDL documents, which describe the protocol bindings and message formats required to interact with the web services listed in its directory.

BPEL [16] defines a notation for specifying process behavior based on interactions of Web services. Web service interactions can be described in two ways: executable processes and abstract processes. Both can be modeled by BPEL. Executable processes model actual behavior of a participant as interactions while abstract processes describe observable behavior and/or process template. BPEL extends the WS-* interaction model to enable business transactions. BPEL defines an interoperable composition model that enable the extension of automated process integration both within and between businesses.

Fig. 3 shows the WS-* workflow as well as the protocol stack. Let us first look at the protocol stack. One may first notice that HTTP performs as transport protocol at the lowest level. Above that, SOAP handles the interaction between services. WSDL and UDDI concern the description and discovery of services at the next higher level. BPEL actually deals with the composition of services at the highest level. Now we look at how these technologies work in a WS-* workflow. Suppose all the available services have registered in the Service Registry. Service Requestor sends a service lookup request described by WSDL to Service Registry. If a suitable candidate service is found, its description is returned to the Service Requestor. Then Service Requester and Service Provider establish connectivity and communicate with each other using SOAP according to the description.

The use of WS-* for smart things dates back many years ago. A Service-Oriented Device Architecture (SODA) [17] is proposed to integrate a wide range of physical devices into distributed IT enterprise systems. In SODA, all the sensors and actuators are exposed as abstract business Web services to the programmers. A bus adapter locates in the boundary between the cyber-world and physical world realms and talks to proprietary and standard device interfaces but presents an uniform Service-Oriented Architecture (SOA) services. Pintus et al. [18], [19] also propose a SOA framework where smart things are described using WSDL standard and logical connections between smart things are modeled as web services orchestrations using the BPEL language. The SOA approach for networks with embedded systems can be also found from many other projects, such as SIRENA [20] and SOCRADES [21], [22].

2) RESTful Architecture: REpresentational State Transfer [23], [24], which was first coined by Roy Fileding in his PhD thesis [25], is considered as the "true architecture of the Web". The basic concept of REST is that everything is modeled "resource", or particularly HTTP resources, with a Universal Resource Identifier (URI). The REST architectural style is based on the following four principles [26]:

- Resource identification through URI. All the resources exposed by RESTful web services are identified by URIs. Through URI, the clients can identify their interaction targets. A global addressing space is provided for service and resource discovery.
- Uniform interface. RESTful services treat the HTTP as an application protocol instead of a transport protocol in WS-*. Therefore, the term REST is often used in conjunction with HTTP and the RESTful resources can be manipulated using HTTP verbs such as PUT, GET, POST and DELETE. PUT creates a new resource while DELETE deletes it. GET retrieves the current state of a resource in some representation while POST updates a resource with new state.
- Self-descriptive messages. Resources are decoupled from their representations such that it is free to use a variety of data formats to describe themselves provided that the appropriate representation formats are agreed and understandable by endpoints. For example, the data can be in any common-used formats such as HTML, XML, plain text, PDF, and

TABLE I.
COMPARISON BETWEEN WS*- AND REST

	WS-*	REST
HTTP	Transport protocol	Application protocol
Complexity	High	Low
Stateless	No	Yes
Mashup	No	Yes
Coupling	Tight	Loosely
Flexibility	Low	High
Security	Built-in	Self-defined

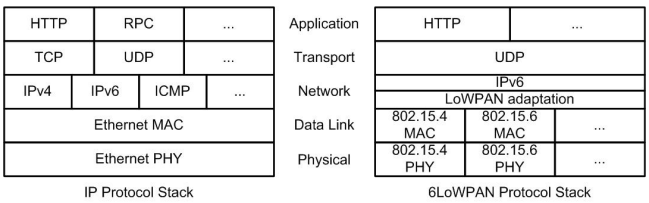

Figure 4. IP and 6LoWPAN protocol stacks

JPEG. Metadata about the resource can be used to control caching, detect transmission errors, negotiate the representation format, and perform authentication or access control between endpoints.

- Stateless operations. Every interaction with a resource itself is stateless. However, stateful interactions can be realized through hyperlinks. The state of a resource can be explicitly transferred by URI rewriting, cookies, and hidden form fields. The states can be also embedded in a response message for stateful interactions.

Notice that although REST is initially described in the context of HTTP, it is not limited to that protocol. RESTful architectures can be based on any other application layer protocols if they can provide a rich and uniform vocabulary for applications to transfer meaningful representational states. By this way, the potential of existing well-defined network protocols can be reexploited without additional efforts.

To our best knowledge, the RESTful architecture is preferred for WoT mainly for its two features. One is its low complexity and the other is its loose-coupling stateless interactions. The two features enable web servers in the RESTful architecture to be embedded into resource-constrained devices (e.g. Resource-oriented architecture [6]) and also enable easy composition (i.e. mashup) of web services. For example, according to [26], REST is the architecture of choice for tactical, ad hoc integration over the Web (i.e., mashup). The previous work on integrating sensor networks to the Internet, [27], [28], has shown that the lightweight aspect of REST makes it an ideal candidate for resource-constrained embedded devices to offer services to the world. To support this opinion, the feasibility of using RESTful web services is demonstrated in [29] with an evaluation of performance and power consumption in an IP-based multi-hop low-power sensor network. More innovative work [6], [11], [29]–[32] applies REST to smart things to abstract them into RESTful web resources mainly under the consideration of both complexity and mashability.

3) Comparison of WS- and REST:* We compare some characteristics of the two different web service paradigms as summarized in Table I.

As analyzed, REST is a more desirable web service paradigm for WoT. However, as indicated in [33], such a resource-oriented approach should not be universally considered as the miracle solution for every problem. In particular, scenarios with very specific requirements, such as high performance real-time communications, might benefit from tightly coupled systems based on different system architectures. While at the same time, for example, once interfaces are quickly created in a target programming language, they can be exposed via WSDL and are consumed just easily if the same WSDL is used by the consumer. Therefore, whether WS-* paradigm or REST paradigm shall be adopted by WoT is still a contradictory issue and we think that the two paradigms will coexist under considerations such as device capabilities and application requirements.

IV. ENABLING TECHNOLOGIES

In this section, we systematically describe and analyze the enabling technologies, including standardization activities as well as existing Web service technologies, which are directly related, or indirectly related but quite important, to WoT.

A. 6LoWPAN

To enable embedded web server on devices, the devices must be addressable, or IP-enabled at first. According to the IP for Smart Objects (IPSO) Alliance, an increasing number of embedded devices will support IP protocol. Many physical objects in the future may be directly connected to the Internet. This trend poses new opportunities for pervasive computing and the Internet. However, new challenges are also introduced. As the things are of different sorts, such as sensors, healthcare devices, RFID tags/readers, and home appliances, the protocol stack should be adaptable to devices with different and limited capabilities, i.e. low memory and low computability.

Responding to the increasing interest of connecting those resource constrained devices to the Internet, the IETF has proposed standards that enable IPv6-based networks. The IETF work group has launched a project called 6LoWPAN, which is an acronym of IPv6 over Low power Wireless Personal Area Networks. It defines encapsulation and header compression mechanisms that allow IPv6 packets to be sent to and received between resource constrained devices usually by adopting low-power radio communication protocols such as IEEE 802.15.4, 802.15.6 or power line communication.

Fig. 4 shows the IPv6 protocol stack with 6LoWPAN in comparison with a typical IP protocol stack. We notice that 6LoWPAN inserts an adaptation layer between the data link layer and the IP layer. The IP communication is

provided above the adaptation layer. The necessity of the adaptation layer is mainly because one IP packet may not fit within one layer 2 frame , e.g. 802.15.4 MAC frame.

The adaptation layer is the main component of 6LowPAN as it enable IPv6 packets to fit into IEEE 802.15.4 frame payload. It has the following functions.

- Header compression. TCP/IP headers are too large to the data link layer protocol, e.g. IEEE 802.15.4, for most devices. For example, IPv6 header has 40 bytes. Without header compression, it is impossible to transmit any payload effectively by a data link layer protocol such as IEEE 802.15.4, which has a maximum packet size of only 128 bytes. By header compression, the header overhead is much reduced. In the best case, the compressed 6LowPAN/UDP header for local unicast communication can be compressed to only 6 bytes while traditional IPv6/UPD header requires 48 bytes.
- Packet fragmentation and reassembling. The data link layer supports packets in small size. For example, IEEE 802.15.4 supports Maximum Transmission Unit (MTU) in size of only 128 bytes while IPv6 packet can be as large as 1280 bytes. This mismatch has to be handled by fragmentation and reassembling in the adaptation layer.
- Edge routing. To connect personal area networks to the Internet, edge routers, which locate on the edge between personal area networks (PANs) and the Internet, play an essential role as they route IP packets into the PAN devices from outside and vice versa. While at the same time, the edge routers also have management features such as distribution of IPv6 prefix and neighbor discovery.

Compared to traditional IP stack, the network layer is limited to IPv6 because IPv4 has reached its exhaustion recently. When a large number of, maybe in trillions, devices need IP addresses, IPv6 is able to make all devices addressable at the IP layer. Although both TCP and UDP are supported, the most common transport protocol used by 6LoWPAN is UDP. The Web can be viewed as the most popular application protocol. Web applications today mainly depend on payloads of HTML, XML, or SOAP carried over HTTP and TCP. The payload can be in size from hundreds of bytes to several KBs, which is too large for use on some 6LoWPAN nodes. Furthermore, the Web applications over 6LoWPAN shall make use of UDP for performance, efficiency and complexity reasons. Therefore, the Web applications over 6LoWPAN shall be fault tolerant due to the unreliability of UDP.

B. CoAP

In 2010, the IETF established a new working group focusing on Constrained RESTful Environment (CoRE). CoRE is characterized by its additional constraints compared to traditional IP networks. To handle those differences and tackle various challenging issues, the CoRE working group is working on a framework for applications

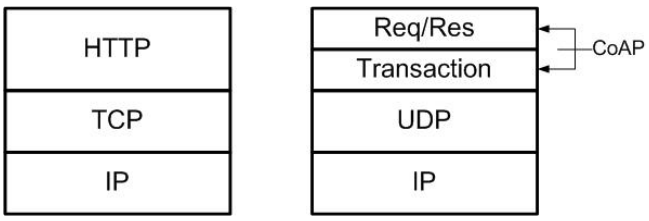

Figure 5. HTTP and CoAP protocol stacks

intended to run on the constrained networks. The framework is designed for applications such as smart energy, home appliance, industry automation, as well as other M2M applications that deal with manipulation of various resources on constrained networks, e.g. monitoring, control and management of resources. As a main part of the framework, Constrained Application Protocol(CoAP) is defined by the CoRE working group.

CoAP can be viewed as a complementary to HTTP as HTTP targets for traditional IP networks such as ethernet while CoAP targets for resource constrained networks such as wireless sensor networks. However, the CoAP protocol can also operate over traditional IP networks. CoAP could be viewed as a compression or redesign of HTTP by taking power, memory and computation constraints into account. Just like HTTP which is designed as transfer protocol for traditional web media content, CoAP is redesigned as a transfer protocol for devices to realize interoperations. The CoAP and HTTP protocol stacks are illustrated in Fig. 5.

The CoAP has the following main features:

- CoAP uses a two-layer approach[4]. Transaction layer is used to deal with UDP and the asynchronous interactions. There are four types of message defined at this layer: *Confirmable*(CON, the message requires acknowledgement), Non-Confirmable (NON, the message does not requrie acknowledgement), Acknowledgement(ACK, it is an acknowledgement to CON), and Reset(RST, the message indicates that a Confirmable message was received, but some context is missing to properly process it). The Req/Res layer is responsible for the transmission of requests and responses for the resource manipulation and interoperation. CoAP supports four request methods: GET, PUT, POST and DELETE, which are answered by a subset of HTTP compatible response codes (e.g. 200 = OK).
- CoAP is based on UDP while HTTP is based on TCP. This is because the high overhead introduced by TCP mechanism such as flow control which is not suitable to resource constrained devices and LLNs. However, CoAP also provides an optional reliable transmission even without the support of TCP. Recall that the *CON* message will be retransmitted if ACK is not received when a predetermined retransmission timer times out. The exponential back-off mechanism

[4]http://tools.ietf.org/html/draft-ietf-core-coap-03

TABLE II.
COMPARISON BETWEEN COAP AND HTTP [34]

	Bytes per-transaction	Power	Lifetime
CoAP	154	0.744 mW	151 days
HTTP	1451	1.333 mW	84 days

is used in retransmissions to avoid congestion. Furthermore, the use of UDP also introduces another benefit that enables best-effort multicast of CoAP while TCP-based HTTP does not support multicast.

- CoAP is designed to lower the header overhead and parsing complexity in order to be applied to resource constrained devices. CoAP uses a short fixed-length compact binary header of only 4 bytes followed by a compact binary option. A typical request has a total header overhead of about 10-20 bytes. A small header overhead avoids the frequent fragmentations. The authors in [34] make a comparison between CoAP and HTTP in terms of average transaction size in bytes, power consumption, and the expected battery lifetime, as shown in Table II. Obviously, traditional HTTP transaction is 10 times bigger than a CoAP transaction. The bigger transaction also results in intensive computation and communication, and consequently higher power consumption, which further shortens the battery lifetime.
- CoAP supports asynchronous transaction, which is a key requirement for M2M applications. When a request can not be responded immediately, the server first acknowledges the reception of the message and sends the response back in an off-line fashion, without risking the client to repeatedly retransmit the request.
- CoAP supports URI and built-in resource discovery. URI is an important feature of the web architecture as the resources must be identified and addressable so as to be searchable and accessible. Resource discovery is common on web. CoAP defines a built-in resource discovery format which allows both discovering and advertising the resources offered by a device.
- CoAP supports a built-in subscribe/notify push model for an end-point to notify another end-point about a resource of interest. In M2M application, it is inefficient for a client to poll whether a resource has changed or not in a pull model. Instead, CoAP provides a built-in push model where a subscription interface is provided for client to request a response whenever a resource changes. This push is accomplished by the device with the resource of interest by sending the response message with the latest change to the subscriber.

For the more detailed specification of CoAP, one may refer to [35]. Although CoAP is still working in progress, some famous embedded operating systems, Tiny OS[5] and Contiki[6], have already released their CoAP implementations. In addition, there are two open source implementations not specially designed for WSNs: one called *libcoap*[7] implemented in C language and the other called *CoAPy*[8] in Python language. A Firefox extension called *Copper*[9] that handles CoAP is also released.

C. Embedded Web Server

For the aforementioned integration of smart things into the Web, either directly or indirectly, embedded web server is indispensable. With embedded web servers, data can be transmitted between the smart things with standard web language. Traditional web servers are mainly designed for high-end computers such as workstations with plentiful CPU and memory resources. For most resource-constrained devices, the embedded web server must be small in footprint and of low complexity in processing while providing web server functionalities as many as possible, e.g. SSL. The different requirements make the traditional web servers unapplicable. Actually, pioneer work has focused on this topic for many years, even before the emergence of the IoT or WoT concept.

In [36], the authors declare that many embedded Internet devices (EID) will use HTTP instead of providing a user interface through a local front panel. They design a web server for EID which doest not require file system and doest not incur the memory and performance overhead either. Furthermore, the solution also provides interoperability between devices. Agranat in [4] shows that devices with a few KB of RAM and EEPROM are able to handle an embedded Web server because efficient TCP/IP implementation adds as little as 48K ROM and 16K RAM extra memory requirements. Can Filibeli et al. [37] design and implement an embedded web server-based home appliance network prototype system where Ethernut-based web servers are embedded into home appliances. With the help of embedded microcontrollers, the home appliances can be controlled and managed via web pages using regular web browsers. Ethernut[10] is an open source hardware and software project for building tiny embedded ethernet devices. It adopts an open source implementation of a real time operating system called Nut/OS and a TCP/IP protocol suite named Nut/Net. It has small footprint, standard C libraries and cooperative multithreading. Priyantha et. al [38] present an approach of implementing the web server on sensor nodes using only 15.8KB ROM and less than 1KB RAM . Duquennoy et al. [2], [39] propose cross-layer approaches to design efficient tiny embedded web servers and implement a prototype, named Smews, which is in size of 7KB and requires only 200 bytes volatile memory. It has been demonstrated that smart cards can be also embedded with web servers. A card with Java-based web server, called serverWebcard [40], is implemented with a TCP/IP stack

[5]http://www.tinyos.net/
[6]http://www.sics.se/contiki/
[7]http://sourceforge.net/projects/libcoap/
[8]http://coapy.sourceforge.net/
[9]https://addons.mozilla.org/en-US/firefox/addon/copper-270430/
[10]http://www.ethernut.de/

and a minimal set of HTTP1.0 functions. OMA(Open Mobile Alliance) specifies Smart Card Web Server (SCWS) standard to allow web servers to be used within smart cards such that network operators' services can be provided through web browser [41]. More importantly, SCWS is portable across any handsets with browsers.

There are much more work than those mentioned above, which implement lightweight embedded web servers with different features in different programming languages. Further efforts are on more powerful embedded web servers but with little resource requirement. It can be expected that the embedded servers will be common in future embedded devices.

D. Service Composition Development

With the emergence of IoT, huge numbers of embedded devices with various functions will be connected to the Internet. Although the connectivity allows devices to provide some specified services on the Internet, it is not the ultimate goal. To fully explore the potential of those devices, they shall be able to cooperate with each other. However, there is a tight coupling among the devices, the services provided as well as the development methods. It is not easy to integrate devices from different manufacturers. While, up to date, new applications in this field are mainly produced by designers and engineers, we claim that even users could invent new applications unforeseen by technical experts with simple and effective composition rules and easy-to-use building blocks.

Fortunately, as we have known, web servers are possible to be embedded into devices such that they can provide web services on the Internet. Even for those which can not directly provide, a smart gateway can act as a proxy to provide web service. Imagine we have several different devices such as temperature sensor, humidity sensor, GPS device as well as some healthcare devices (e.g. EKG sensor). We want to build a healthcare system which can monitor and record the health condition of patients as well as environment information (e.g. temperature, humidity, position) of the patient. Traditionally, the developer shall understand all the native APIs provided by each device and write programs to integrate information provided by those devices. This requires extensive time and technical expertise. In WoT, all the things are abstracted as web resources, which are addressable, searchable and accessible on the Web. The developer can use uniform web standard to integrate all the abstracted web resources needed as well as existing virtual web service (e.g. Google Map) together to create a mashup. Mashups are new web application/service created by composing various original web services from disparate, or even competing providers. Mashup can be viewed as a key feature of Web 2.0 or one of the main differences to Web1.0. In Web2.0, There are plentiful web service available such as Twitter, Facebook, Flickr, Linkedin, eBay, Yahoo Maps, and so on. Most provide APIs to allow developers to create more new services based on their basic services. This has become a major web application development trend. For example, a web service, called Wikipediavision[11], is created by using Google Map APIs and Wikipedia APIs to timely show the places where anonymous edits to Wikipedia are happening on Google Map. However, different from traditional Web2.0 mashup, mashups in WoT include not only virtual web services but also physical web services provided by things. Some researchers call this Web3.0 mashup and argue that we are going to enter a Web3.0 era. As more web-enabled things will be abstracted as web services and published on the Web, together with the popularity and progress of virtual world web services, more fruitful and powerful composite Web3.0 services can be envisioned in the future. We believe that Web3.0 mashup will be the main technical engine to make progress for both WoT and IoT.

In [6], Guinard et al. apply REST principles to embedded devices and present two representative Web3.0 mashup styles, physical-virtual mashups and physical-physical mashups.

1) Physical-Virtual Mashup: As indicated by its name, this mashup consists of web services from both the physical world and the virtual world. Although the embedded devices can provide some web services to answer HTTP queries from users such as checking the state of the devices or changing the state of the devices, it might not always be sufficient to satisfy the user's requirement. For example, suppose some sensor nodes are distributed over the city to monitor the temperature of different spots. It is desirable that the values can be displayed in a visual way (e.g. on a map) such that people can easily get the information about any specified spot. Under such requirement, the developer can mashup virtual map service and physical sensor web service to create a temperature monitoring web application which displays temperatures of different places on the map. Any services, either physical or virtual, are able to be mashuped if they follow the same standard (e.g. REST) and provide an uniform interface to communicate with.

Many mashup products or prototypes have been developed including both virtual and physical services. The WoT example shown in Section I can be viewed as a classical physical-virtual mashup. Guinard et al. [6] implement an application, called EnergyVisualizer, which offers a GUI on the Web to monitor the power consumption and to control different home appliances. EnergyVisualizer is built by using the self-defined RESTful Plogg API and Google Web Toolkit APIs. The mashup calls the Ploggs Smart Gateway at a constant interval by issuing a GET HTTP request to the Ploggs and feeds the response in an interoperable data in JSON format to the corresponding graphs. Furthermore, they also put switch buttons on the web page, where by clicking a button the corresponding appliance can be turn on or off. In the cloud computing industry, the providers, e.g. Amazon Web Services, Google's Google Apps, and Salesforce.com's Force.com., use web interface or API to allow users to provision and scale physical servers,

[11] http://www.lkozma.net/wpv/index.html

storage, networking, load balancing and security in real time and in multiple data centers. The systems generally use SOAP, WSDL, and other nonproprietary XML-based web service protocols.

2) Physical-Physical Mashup: As what the term indicates, the mashup consists of web services only from the physical world. In this kind of mashup, the original web services are all provided by smart things, either directly or indirectly. It enable devices with various functionalities from different manufacturers or even competitors to cooperate with each other (e.g. a humidity sensor from one vendor controlling a sprinkler system from another). The developers do not require expert knowledge about the programming methods and tools about each device as they also have been abstracted as web resources. A uniform and interposable web API can be used to communicate with all of them.

Guinard et al. [6] demonstrate how physical-world services can be combined together using mashup technologies. They implement an Ambient Meter on a Sun SPOT which polls a predetermined URL using GET method to get the energy consumption of all the devices in a room from a smart gateway. All the devices communicate with HTTP-based requests and responses. They find that it would be much time consuming if the smart gateways, the Ploggs and the Sun SPOTs only offer their native APIs. Using the same concept as [6], Kamilaris et al. [42] develop an energy-aware/cost-aware smart home platform. Besides integrating the services to provide the energy consumption about the home appliances, they further integrate the smart grid web services to provide the real-time tariff. The composite service allows residents to save energy as well as money by defining rules through the Web.

V. Open Platforms and Prototypes

In this section, we list some open platforms and ptotypes that have been implemented and presented in the literatures. Most of those platforms have been available on the Internet and accessible by web browsers. Web service APIs are also provided such that users can use them to create more innovative applications or services.

SenseWeb [43], [44] is developed at Microsoft Research. It offers a platform mainly targeting for particpatory sensing. A sensor gateway is used by sensors as a uniform interface to share sensory data. SOAP-based APIs are used to allow developing sensing applications with shared sensing resources. For example, SensorMap [45] is one such application. It mashes up sensor data from SenseWeb on a geographical map interface. In particular, it enable selective sensor queries and data visualization. Also the access and management of sensors are conducted in an authentical way. Nath et al. [45] point out that to realize the full potential of a portal like SensorMap, it should be easily extensible and mashed up with other applications and services. They are currently working on a set of modulars and composable APIs to facilitate mashing up SensorMap with other services.

SensorBase[12] [46] implemented in the Center for Embedded Networked Sensing (CENS) at UCLA uses a relational database table as its data abstraction and SQL-centric APIs. It is a web application that not only provides the user with the functionality of a traditional database management system, but also runs under the notion of a Web 2.0 data experience with a responsive user interface design and RSS data feed techniques.

Sensorpedia[13] [47] is a web-based application developed at Oak Ridge National Laboratory, enabling people to share, find, and use sensor data online. It provides users a Google Maps interface where users can search and explore published sensor data. Sensorpedia applies several design principles common to many popular Web2.0 sites. The Sensorpedia APIs allow accepting and publishing data by established standards such as the Atom Syndication Format. The APIs also support rapid development of customized third-party applications to meet specific user requirements.

Sensor.Network [48], [49] implemented by Gupta et al. in Sun Microsystems is a Web-based infrastructure for storing, sharing, searching, visualizing and analyzing data from heterogeneous devices. Interactions amongst devices or with end users are through an open REST-base API. They also propose a category-based search mechanism and security mechanisms for authentication, authorization and confidentiality.

Pachube [14] is a venture capital funded data brokerage platform for IoT, managing millions of data points per day from thousands of individuals, organizations and companies around the world. Pachube provides APIs entirely based on HTTP requests, and conforms to the design principles of REST. The "physical-to-virtual" APIs provided by Pachube enable quick and easy development of applications that add value to networked objects and environments. The WoT example, Japan Geigermap, shown in Section I is built by Pachube service and Google Maps service.

Vazquez et al. [50] propose Flexco, which is a flexible architecture for implementing monitoring applications based on wireless sensor networks. They propose a three-layer architecture (i.e. Sensors and Actuators Layer, Coordination Layer and Supervision Layers), which enables intelligence distribution and decision at different levels. On its top Supervision layer, a web interface is proposed for end users to access and manage the sensor data.

TinyREST architecture is proposed in [27], where the authors implement a prototype using MICAz motes. Especially, they introduce a new HTTP method, SUBSCRIBE, which enables clients to register their interests to specific sensors/actuators services with various personalized parameters depending on each client's needs. Also, a multithreaded light-weight HTTP-2-TinyREST gateway is provided between clients and sensors/actuators.

The pico-REST (pREST) [30] is an access protocol

[12] http://sensorbase.org/
[13] http://www.sensorpedia.com/
[14] http://www.pachube.com/

proposed by Drytkiewicz et al. with the goal of bringing the Web simplicity and a holistic view on data and services to pervasive systems. In a REST style, pREST emphasizes abstraction of data and services as resources. A particular concern is to provide the functionality in the absence of proxy nodes or infrastructure services like directory servers.

The EnergieVisible [6], [51] software can be used to easily monitor the power consumption of devices connected to Bluetooth-enabled smart plugs (Ploggs). The software retrieves all Ploggs in the environment through a smart gateway using REST APIs and exposes their functionalities (i.e. power consumption data and an on/off switch) via a visualized web page.

Table III summarizes and compares some work mentioned above from different aspects. The work whose features are unknown is not listed in the table.

VI. Open issues

To fully explore the potential of WoT, many challenging issues still need to be tackled. In this section, we review some open issues.

A. Heterogeneity and Scalability

Although WoT is a good approach to handle the heterogeneity problem, some minor heterogeneity problems of devices and requirements still exist.

The popularity of WoT requires a tremendously huge number of devices to be integrated to the existing Web. These devices are diverse in terms of data communication methods and capabilities (e.g., protocol stack, data-rate, reliability, etc.), computational and storage power, energy availability, adaptability, mobility, etc. The heterogeneity at the device level seriously challenges to the popularity of WoT. WoT is the concept of standardizing communication channel at the application layer. Without the interoperation support from lower layers, WoT is just a castle in the air. This issue is still under investigation, but it is hard to find a one-fit-all solution as new devices may appear in the future.

On the other hand, consumers of data are heterogeneous: someone might ask for realtime information while some others might need archived data streams from the past. Their needs vary in terms of data quality, spatial resolution, and sampling rates. Further, different applications might implement disparate data processing or filtering. WoT shall be open to support these various applications whose characteristics and requirements may be extremely diverse, in terms of bandwidth, latency, reliability, etc. These heterogeneity traits of the overall system make the design of a unifying framework and the communication protocols a very challenging task, especially with devices with vastly different levels of capabilities.

In addition, management of WoT becomes very difficult in a large distributed environment, and solutions to dominate the complexity need to be found. Without a careful management mechanism design, it might result in an inevitable performance degradation. The power of WoT comes from the growth of participant number of devices. A management mechanism shall be able to distinguish both the functionalities and the capabilities of devices. Otherwise, it can not specify or allocate appropriate devices and services for user application requirements. It is nontrivial to manage growing number of devices in a graceful manner, especially when the devices are heterogeneous in functionalities and capabilities. On the other hand, the resource-constrained devices, although are able to be embedded with web servers, are still limited to handle large number of requests. Unlike a stand-alone system, one device is shared by a small number of applications. In WoT, it is hard to say how many application requests need to be handled by a device, especially when it provides public services. To keep the resource usage scalable and to avoid unnecessary denied accesses to some applications, it is essential to design mechanisms that can coordinate the smart things under the consideration of both their capabilities and user application requirements. Also, it is possible to improve the scalability by more advanced embedded web server techniques.

B. Security and Privacy

Openness and sharing are always contradictory to security and privacy. One practical consideration in enabling widespread adoption of WoT arises in ensuring security of shared resources against misuse, protecting the privacy of users who share parts of their data, and providing estimates of reliability or verifiability of web service against malicious intervention or inadvertent errors. Although the security and privacy have been extensively studied for decades and some techniques have become mature, not all existing technologies can be directly applied to smart things in WoT. The problem is exacerbated by introducing large-scale, distributed, heterogeneous and low-capability smart things.

For security, the CoRE working group has been exploring approaches to security bootstrapping that are realistic under the given constraints and requirements of the network. To ensure that any two nodes can join together, all nodes must implement at least one universal bootstrapping method. Security can be achieved using either session security or object security. Cipher suite will also be redesigned so as to be implemented with a minimal requirement. In [52], the author presents an analysis of security threats to the 6LoWPAN adaptation layer from the point of view of IP packet fragmentation attacks and proposes a protection mechanism against such attacks using time stamp and nonce options that are added to the fragmentation packets at the 6LoWPAN adaptation layer.

Allowing the information available on the Web poses a perceived privacy threat. The approach to use existing authentication service from third parties has been advocated. For example, Sensorpedia [46] relies on open data portability standards such as OData[15], oEmbed[16],

[15]http://www.odata.org/

[16]http://www.oembed.com/

TABLE III.
A BRIEF COMPARISON OF EXISTING OPEN PLATFORMS AND PROTOTYPES (PART OF THE TABLE REFERS TO [49])

	Sensor.Network	SensorBase	Pachube	SenseWeb	TinyREST	pREST	EnergieVisible
Integration	Hybrid	Hybrid	Hybrid	Hybrid	Indirect	Direct	Indirect
Web service paradigms	RESTful	WS-*	RESTful	WS-*	REST	REST	REST
Data formats	XML, JSON	XML, JSON	JSON	Text	Unknown	XML	JSON
Architecture	Centralized	Centralized	Centralized	Centralized	Distributed	Distributed	Distributed
Interoperability	No	No	No	No	Yes	Yes	Yes

OpenID[17], and OAuth[18] to ensure current and future interoperability with other web-based software applications. Some web service might be shared within restricted groups only. For example, home appliance web service shall be only accessible to family members. Following the idea of leveraging existing social structure on online social networks (OSNs, e.g. Twitter, Faceook, Linkedin, etc.) and their APIs to define the access privilege of smart things, Guinard et al. implement a prototype, called Social Access Controller(SAC), which is an authentication proxy between users and smart things. OSN-based methods can handle the access control between people and things but are unable to deal with the access control between things. Universal but distributed access control mechanism is expected to enable interoperation between things while preserving the privacy of the owners.

C. Search and Discovery

Both people and things may need to discover the existence, functionality and information of their desired web services. For example, things require identities of smart things and web services within their environment in order to negotiate about shared goals to create a new mashup according to some requirement. Search engine is essential to WoT. Generally, as indicated in [53], there are two fundamental approaches to construct a search engine for WoT. In the push approach, sensor outputs are proactively pushed to a search engine, which uses the data to resolve queries reactively. However, this method lacks of scalability in the smart things-based crowd-sourcing environment. It can be only applied to a system with limited number of devices. Alternatively, in the pull approach, only upon receiving a user query, the search engine forwards it to the sensors to pull the relevant data. This method is scalable but challenged by the accuracy and timeliness. Here, we focus on the latter one.

The increasing penetration of Web with smart things leads to more crowd-sourcing than ever before. A large amount of information and physical world web services of various sorts become available on the Web. Although more services may be beneficial and convenient to people, it becomes a nightmare to the search engine of WoT. The search engine is already not an easy issue in Web2.0 crowd-sourcing with a mass of web contents created everyday, not to mention Web3.0 crowd-sourcing introduced by trillion of smart things.

Furthermore, a key service for WoT will be the search engine that allows to search a physical-world service with certain properties. The traditional Web is dominated by static or slowly changing contents that are manually typed in by humans. The contents in WoT are rapidly changing because they are automatically produced by smart things. Thus, a search engine for WoT shall support searching rapidly changing content. This is a key challenge because existing search engines are based on the assumption that most web contents change slowly such that it is sufficient for the search engine to update an index at a low frequency. This is clearly impossible for the WoT where the states of many physical world devices changes are at frequency of minutes or even seconds. On the other hand, some Web content or service is significant only during a specified duration. In addition, future mashup shall be created dynamically on-demand according to the context. The source web services may need to be searched and obtained dynamically and in realtime. This issue becomes more challenging due to the dynamics of WoT, introduced by its features such as mobility and intermittent connectivity of smart things. The search engine for WoT shall support real-time search of information and real-time discovery of web services.

There has been some pioneer work on this issue. Ostermaier et al. [53] show how the existing web infrastructure can be leveraged to support publishing of sensor and entity data. They implement a prototype of real-time search engine, called Dyser, which enables finding the real-world devices that exhibit a certain state at the time of the query. In [54], the authors survey and clarify relevant existing approaches (e.g. Snoogle [55], Microsearch [56], MAX [57], etc.) according to query type, language, scope, accuracy and so on. Mayer et al. [58] present DiscoWoT, a semantic discovery service for Web-enabled smart things. DiscoWoT is based on the application with multiple discovery strategies to a representation of web resource, where arbitrary users can create and update the strategies at runtime using DiscoWoT's RESTful interface.

D. Ambient Intelligence

The ultimate goal of IoT or WoT is to build an ecosystem that can provide user-oriented and environment-aware services. In other words, the web services shall be sensitive and responsive to the presence of people and the condition of environment. Ambient Intelligence(AmI) has been much addressed on stand-alone systems, such as wireless sensor and actuator networks. The sensor capable of recognizing simple emergency situation may fire an

[17] http://openid.net/
[18] http://www.oauth.net/

alarm and the actuator can take an action accordingly. When it comes to AmI in WoT, new opportunities and challenges are exposed. The community effect of the web services available on a larger-scale Web shall be further addressed. One may easily find different public services on the Web and build private web services using standard web-enabled devices in personal area network. The challenges first come from the heterogeneity and availability of smart things that provide web services. Unlike standalone systems where the devices are predetermined and configured according to the application requirement, some AmI applications in WoT may need to discover the required web services first. The QoS, even the existence, of the web service is unknown. Furthermore, this situation is exacerbated by the unexpected user requirements and environment (e.g. time, location, etc.).

Recall that mashup technology is a key enabling technology of WoT. It can be expected that the mashups will be dominant in WoT. Another challenge of AmI in WoT is that the mashup shall intelligently adapt to user requirements and runtime environment. In other words, it shall be context-aware. In such a condition, dynamic mashup could be a good option. Other than developing static mashup by integrating existing web services together, rules about how to mashup services should be defined such that the basic web services are dynamically added or deleted on-demand. The whole mashup processes are transparent to the users and without human intervention. For example, to build a healthcare system for the elderly requires some private services to monitor and record their health conditions and some public services to know the environment information (e.g. temperature, humidity, light, traffic, etc.) about the places where they locate. The public services to be integrated shall be dynamically chosen according to their positions. In an emergency condition, some services shall be automatically activated and integrated, e.g., the control service for automatic syringe might be activated and responded accurately according to the health condition and the environment parameters known from the other services.

AmI of WoT is far more powerful and sophisticated than those examples. To fully explore the potential of smart things, more innovative solutions are expected to be proposed. Those solutions shall be able to orchestrate all available web services in a graceful manner and enable more intelligent user-oriented services. Some existing artificial intelligent concepts and technologies, such as Collective intelligence [59] and Semantic web services [60], may deserve revisiting in the hope of finding new efficient solutions feasible to smart things on the Web.

VII. Conclusion

IoT is the next big possibility and challenge of the Internet. It does not merely concern the connectivity of smart things, but more about the interaction or interoperation between things and between things and people. This requires that all the smart things can speak the same language to communicate freely with each other. It has been considered as a good solution to extend existing web architecture to this new domain by incorporating smar things into the Web. The extended Web is called as WoT in the literature and it has become a major trend to promote the development of IoT.

In this paper, we first give an overview of WoT, including the history and motivation, and the comparison with previous technologies (e.g., UPnP, JXTA, etc.). It shows many advantages of WoT over previous technologies. The key concept of WoT is abstracting everything into web service. The first issue to consider is to integrate smart things to the Web by either direct integration or indirect integration, depending on their capabilities. The next issue is how to abstract the integrated smart thing to web services. We introduce and compare two major web service architectures: WS-* architecture and RESTful architecture. Some key enabling standards and technologies (e.g. 6LoWPAN, CoAP, mashup, etc.) related to WoT are also discussed and examined. As examples for case study, we compare some pioneer implementation of open platforms and prototypes for WoT which provide web service APIs for sensory data sharing and device interoperabilities. Since we are still at the preliminary stage of WoT, many open challenging issues are also briefly analyzed. We believe that WoT will be indispensable in people's future lives and more efforts to tackle those challenging issues shall be made from both industry and academia to promote the progress of WoT.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, pp. 2787–2805, October 2010.

[2] S. Duquennoy, G. Grimaud, and J.-J. Vandewalle, "Smews: Smart and mobile embedded web server," in *Complex, Intelligent and Software Intensive Systems, 2009. CISIS '09. International Conference on*, march 2009, pp. 571 – 576.

[3] B. Traversat, M. Abdelaziz, D. Doolin, M. Duigou, J.-C. Hugly, and E. Pouyoul, "Project JXTA-C: enabling a Web of things," in *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, 2003, p. 9 pp.

[4] I. Agranat, "Engineering web technologies for embedded applications," *Internet Computing, IEEE*, vol. 2, no. 3, pp. 40 –45, may/jun 1998.

[5] T. Lin, H. Zhao, J. Wang, G. Han, and J. Wang, "An embedded Web server for equipment," pp. 345 – 350, may 2004.

[6] 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), in proceedings of WWW (International World Wide Web Conferences)*, Madrid, Spain, Apr. 2009.

[7] O. Akribopoulos, I. Chatzigiannakis, C. Koninis, and E. Theodoridis, "A Web Services-oriented Architecture for Integrating Small Programmable Objects in the Web of Things," *2010 Developments in E-systems Engineering*, pp. 70–75, 2010.

[8] B. Ostermaier, M. Kovatsch, and S. Santini, "Connecting things to the web using programmable low-power wifi

modules," in *Proceedings of the 2nd International Workshop on the Web of Things (WoT 2011), San Francisco, CA, USA*, June 2011, accepted for publication.

[9] K. il Hwang, J. In, N. Park, and D. seop Eom, "A design and implementation of wireless sensor gateway for efficient querying and managing through world wide web," *Consumer Electronics, IEEE Transactions on*, vol. 49, no. 4, pp. 1090 – 1097, nov. 2003.

[10] V. Trifa, S. Wiel, D. Guinard, and T. 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)*, 2009.

[11] D. Guinard, C. Floerkemeier, and S. Sarma, "Cloud computing, rest and mashups to simplify rfid application development and deployment," in *Proceedings of the 2nd International Workshop on the Web of Things (WoT 2011)*. San Fransisco, USA: ACM, June 2011.

[12] W3C Working Group, "Web Services Architecture," http://www.w3.org/TR/ws-arch/.

[13] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer, "Simple object access protocol (SOAP) 1.1," 2000.

[14] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web services description language (WSDL) 1.1," http://www.w3.org/TR/wsdl, 2001.

[15] T. Bellwood, L. Clément, D. Ehnebuske, A. Hately, M. Hondo, Y. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, *et al.*, "UDDI Version 3.0," *Published specification, Oasis*, vol. 5, pp. 16–18, 2002.

[16] D. Jordan, J. Evdemon, A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, *et al.*, "Web services business process execution language version 2.0," *OASIS Standard*, vol. 11, 2007.

[17] S. de Deugd, R. Carroll, K. Kelly, B. Millett, and J. Ricker, "Soda: Service oriented device architecture," *Pervasive Computing, IEEE*, vol. 5, no. 3, pp. 94 –96, july-sept. 2006.

[18] A. Pintus, D. Carboni, A. Piras, and A. Giordano, "Connecting smart things through web services orchestrations," in *Proceedings of the 10th international conference on Current trends in web engineering*, ser. ICWE'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 431–441.

[19] ——, "Connecting smart things through web services orchestrations," in *Proceedings of the 10th international conference on Current trends in web engineering*, ser. ICWE'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 431–441.

[20] F. Jammes and H. Smit, "Service-oriented paradigms in industrial automation," *Industrial Informatics, IEEE Transactions on*, vol. 1, no. 1, pp. 62 – 70, feb. 2005.

[21] L. de Souza, P. Spiess, D. Guinard, M. Khler, S. Karnouskos, and D. Savio, "Socrades: A web service based shop floor integration infrastructure," in *The Internet of Things*, ser. Lecture Notes in Computer Science, C. Floerkemeier, M. Langheinrich, E. Fleisch, F. Mattern, and S. Sarma, Eds. Springer Berlin / Heidelberg, 2008, vol. 4952, pp. 50–67.

[22] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. M. S. d. Souza, and V. Trifa, "Soa-based integration of the internet of things in enterprise services," in *Proceedings of the 2009 IEEE International Conference on Web Services*, ser. ICWS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 968–975.

[23] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," pp. 407–416, 2000.

[24] ——, "Principled design of the modern web architecture," *ACM Trans. Internet Technol.*, vol. 2, pp. 115–150, May 2002.

[25] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, 2000, aAI9980887.

[26] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs. "big"' web services: making the right architectural decision," in *Proceeding of the 17th international conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 805–814.

[27] 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.

[28] S. Mäkeläinen and T. Alakoski, "Fixed-mobile hybrid mashups: Applying the rest principles to mobile-specific resources," in *Proceedings of the 2008 international workshops on Web Information Systems Engineering*, ser. WISE '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 172–182.

[29] D. Yazar and A. Dunkels, "Efficient application integration in IP-based sensor networks," in *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, ser. BuildSys '09. New York, NY, USA: ACM, 2009, pp. 43–48.

[30] W. Drytkiewicz, I. Radusch, S. Arbanowski, and R. Popescu-Zeletin, "pREST: a REST-based protocol for pervasive systems," in *Mobile Ad-hoc and Sensor Systems, 2004 IEEE International Conference on*. IEEE, 2004, pp. 340–348.

[31] E. Wilde, "Putting things to rest," School of Information, UC Berkeley. Report 2007-015., Tech. Rep., 2007. [Online]. Available: http://escholarship.org/uc/item/1786t1dm

[32] V. Stirbu, "Towards a RESTful Plug and Play Experience in the Web of Things," in *Proceedings of the 2008 IEEE International Conference on Semantic Computing*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 512–517.

[33] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, *From the Internet of Things to the Web of Things: Resource Oriented Architecture and Best Practices*. Springer, Dec. 2010, ch. 5.

[34] W. Colitti, K. Steenhaut, and N. De Caro, "Integrating Wireless Sensor Networks with the Web," in *Extending the Internet to Low power and Lossy Networks (IP+SN 2011)*, 2011.

[35] C. B. Z. Shelby, K. Hartke and B. Frank, "Constrained application protocol (coap)," http://tools.ietf.org/html/draft-ietf-core-coap-05, March 2011.

[36] A. Wilson, "The challenge of embedded internet design," *Real-Time Magazine*, pp. 78–80, 1998.

[37] M. Can Filibeli, O. Ozkasap, and M. Reha Civanlar, "Embedded web server-based home appliance networks," *J. Netw. Comput. Appl.*, vol. 30, pp. 499–514, April 2007.

[38] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, "Tiny web services: design and implementation of interoperable and evolvable sensor networks," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, ser. SenSys '08. New York, NY, USA: ACM, 2008, pp. 253–266.

[39] S. Duquennoy, G. Grimaud, and J.-J. Vandewalle, "The Web of Things: Interconnecting Devices with High Usability and Performance," in *Embedded Software and Systems, 2009. ICESS '09. International Conference on*, May 2009, pp. 323 –330.

[40] J. Rees and P. Honeyman, "Webcard: a java card web server," in *Proceedings of the fourth working conference on smart card research and advanced applications on Smart card research and advanced applications*. Norwell, MA, USA: Kluwer Academic Publishers, 2001, pp. 197–207.

[41] O. M. A. Ltd., "Smartcard-web-server," April 2008.

[42] A. Kamilaris and A. Pitsillides, "Exploiting Demand Response in Web-based Energy-aware Smart Homes," in *The*

First International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies, 2011.

[43] A. Santanche, S. Nath, J. Liu, B. Priyantha, and F. Zhao, "Senseweb: Browsing the physical world in real time," *Demo Abstract, ACM/IEEE IPSN06, Nashville, TN*, 2006.

[44] W. Grosky, A. Kansal, S. Nath, J. Liu, and F. Zhao, "Senseweb: An infrastructure for shared sensing," *Multimedia, IEEE*, vol. 14, no. 4, pp. 8 –13, oct.-dec. 2007.

[45] S. Nath, J. Liu, and F. Zhao, "Sensormap for wide-area sensor webs," *Computer*, vol. 40, no. 7, pp. 90 –93, july 2007.

[46] M. H. Gong Chen, Nathan Yau and D. Estrin, "Sharing sensor network data," in CENS Technical Report 71, Tech. Rep., March 2007.

[47] B. L. Gorman, D. R. Resseguie, and C. Tomkins-Tinch, "Sensorpedia: Information sharing across incompatible sensor systems," in *Proceedings of the 2009 International Symposium on Collaborative Technologies and Systems*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 448–454.

[48] V. Gupta, A. Poursohi, and P. Udupi, "Sensor.Network: An open data exchange for the web of things," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, 292010-april2 2010, pp. 753 –755.

[49] V. Gupta, P. Udupi, and A. Poursohi, "Early lessons from building Sensor.Network: an open data exchange for the web of things," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, 29 2010-april 2 2010, pp. 738 –744.

[50] J. Vazquez, A. Almeida, I. Doamo, X. Laiseca, and P. Orduña, "Flexeo: an architecture for integrating Wireless Sensor Networks into the Internet of Things," pp. 219–228, 2009.

[51] D. Guinard, M. Weiss, and V. Trifa, "Are you energy-efficient? sense it on the web!" in *Adjunct Proceedings of Pervasive 2009 (International Conference on Pervasive Computing)*, Nara, Japan, May 2009.

[52] H. Kim, "Protection against packet fragmentation attacks at 6lowpan adaptation layer," in *Proceedings of the 2008 International Conference on Convergence and Hybrid Information Technology*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 796–801.

[53] B. Ostermaier, K. Römer, F. Mattern, M. Fahrmair, and W. Kellerer, "A Real-Time Search Engine for the Web of Things," in *Proceedings of Internet of Things 2010 International Conference (IoT 2010)*, Tokyo, Japan, Nov. 2010.

[54] K. Römer, B. Ostermaier, F. Mattern, M. Fahrmair, and W. Kellerer, "Real-time search for real-world entities: A survey," Nov. 2010.

[55] H. Wang, C. Tan, and Q. Li, "Snoogle: A search engine for pervasive environments," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 8, pp. 1188 –1202, aug. 2010.

[56] C. C. Tan, B. Sheng, H. Wang, and Q. Li, "Microsearch: When search engines meet small devices," in *Proceedings of the 6th International Conference on Pervasive Computing*, ser. Pervasive '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 93–110.

[57] K.-K. Yap, V. Srinivasan, and M. Motani, "Max: human-centric search of the physical world," in *Proceedings of the 3rd international conference on Embedded networked sensor systems*, ser. SenSys '05. New York, NY, USA: ACM, 2005, pp. 166–179.

[58] S. Mayer and D. Guinard, "An extensible discovery service for smart things," in *Proceedings of the 2nd International Workshop on the Web of Things (WoT 2011)*. San Fransisco, USA: ACM, June 2011.

[59] P. Lévy, *Collective intelligence: Mankind's emerging world in cyberspace*. Perseus Publishing, 1999.

[60] S. Narayanan and S. A. McIlraith, "Simulation, verification and automated composition of web services," in *Proceedings of the 11th international conference on World Wide Web*, ser. WWW '02. New York, NY, USA: ACM, 2002, pp. 77–88.

Deze Zeng is currently a Ph.D. candidate at University of Aizu, Aizu-Wakamatsu, Japan. He received his BS degree from School of Computer Science and Technology, Huazhong University of Science and Technology, China in 2007 and MS degrees from University of Aizu, Aizu-Wakamatsu, Japan in 2009.

His current research interests are mainly in the areas of protocol design and performance analysis of wireless networks, with a special emphasis on MAC protocol design and Delay Tolerant Networks. His research interests also include Wireless Sensor Networks, Pervasive Computing and Internet of Things.

Song Guo received the Ph.D. degree in computer science from the University of Ottawa, Ottawa, Canada, in 2005. Since then, he held a position with the Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC, Canada, on a Natural Sciences and Engineering Research Council of Canada (NSERC) Postdoctoral Fellowship. He is currently an Associate Professor with the School of Computer Science and Engineering, University of Aizu, Aizu-Wakamatsu, Japan. His research interests are mainly in the areas of protocol design and performance analysis for computer and telecommunication networks, presently focusing on modeling, analysis, cross-layer optimization, and performance evaluation of wireless ad hoc and sensor networks for reliable, energy efficient, and cost-effective communications. He is senior member of IEEE.

Zixue Cheng received the B.Eng. degree from Northeast Heavy Machinery Institute, Qinhaungdao, China, in 1982, and the M.S. and Ph.D. degrees in engineering from Tohoku University, Sendai, Japan, in 1990 and 1993, respectively.

He was an Assistant Professor from 1993 to 1999, an Associate Professor from 1999 to 2002, and has been a Full Professor since 2002 with the University of Aizu, Aizu-Wakamatsu, Japan. His current interests include distributed algorithms, distance education, ubiquitous learning, context-aware service plat-forms, and functional safety for embedded systems.