INPUT CONTRIBUTION

Group Name:*	oneM2M WG5(MAS)
Title:*	Proposal of a cloud-based presentation and application framework for developing and deploying Web of Things Applications
Source:*	KCA, KAIST(TTA)
Contact:	Nam Giang, Minkeun Ha, Daeyoung Kim (KAIST) Wongyu Jang, SungHyup Lee, Kyonggun Kim(KCA)
Date:*	2013-06-18
Abstract:*	This contribution proposes a cloud-based presentation and application framework for building and deploying web applications that use physical things resources.
Agenda Item:*	TBD
Work item(s):	TBD
Document(s) Impacted*	N/A
Intended purpose of document:*	☐ Decision ☒ Discussion ☒ Information ☐ Other <specify>
Decision requested or recommendation:*	N/A

oneM2M IPR STATEMENT

Participation in, or attendance at, any activity of oneM2M, constitutes acceptance of and agreement to be bound by all provisions of IPR policy of the admitting Partner Type 1 and permission that all communications and statements, oral or written, or other information disclosed or presented, and any translation or derivative thereof, may without compensation, and to the extent such participant or attendee may legally and freely grant such copyright rights, be distributed, published, and posted on oneM2M's web site, in whole or in part, on a non-exclusive basis by oneM2M or oneM2M Partners Type 1 or their licensees or assignees, or as oneM2M SC directs.

1.0 Introduction

Web of Things (WoT) is a promising trend in which web data are not only contributed by human but by a large number of connected physical things. However, developing applications and services that use physical things as resources is not an easy task since the developers usually need to know about system level protocols. This document proposes Pretty, a cloud-based presentation and application framework for developing and deploying web applications for the WoT. Leveraging cloud system, Pretty provides a scalable development framework with which developers can easily create and deploy WoT applications and services (WoT A/S) without having to know about system level protocols.

2.0 Constitution and Scope

This document focuses on introducing the designed framework, which includes both architectural design and component design. It also describes some use case scenarios that show how easy it is to create and deploy WoT A/S using the framework. Additionally, a demonstration that shows how end users consume the developed WoT A/S is also included.

3.0 Terms and Abbreviations

CDM: Cross Document Messaging
WS: Web Socket
CoAP: Constrained Application Protocol
RIA: Rich Internet Application
WoT A/S: Web of Things Application/Service

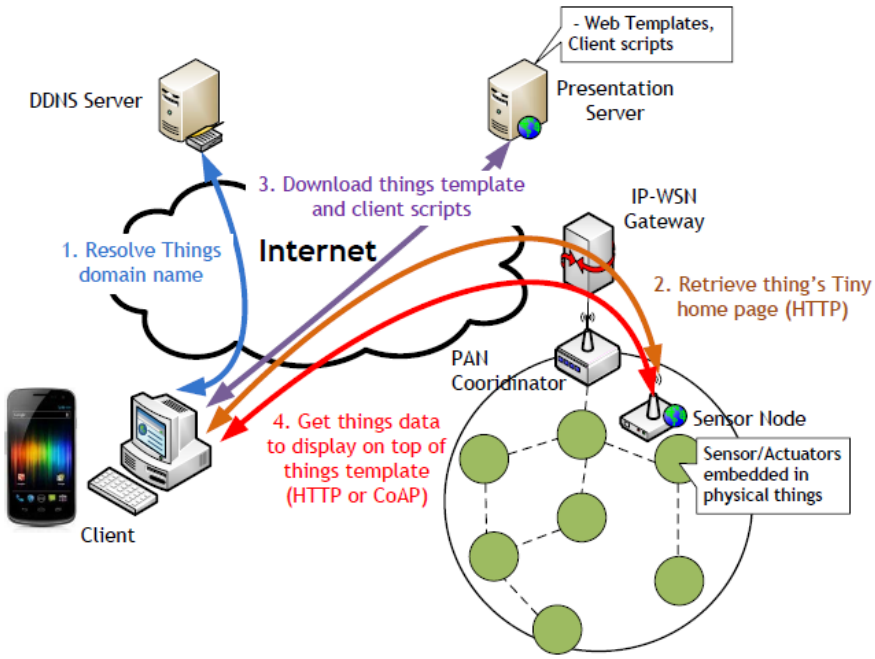
4.0 Overview of Pretty framework

The IoT vision anticipated that there would be a tremendous and continuously growing amount of connected things. This means a huge amount of information will be contributed by physical things. Unfortunately, physical things are too diverse in types and functionalities so that it is difficult in building applications and services that leverage things' resources. In this sense, developers usually need to study and have a thorough knowledge on vast sort of underlying system level protocols instead of just focus on their applications. Moreover, since physical

things are supposed to be connected to the Internet, a web-based development framework that creates rich internet application - RIAs for things would be appropriate. Pretty framework is developed following this motivation. Pretty is a cloud-based framework that offers a development tool and APIs, which help facilitate the process of developing and deploying WoT A/S.

4.1. Service architecture

Fig. 4-1 shows the general theory of developing a RIA that leverages physical things' resources.



(Fig. 4-1) Theoretical architecture for physical things-based web application

In order to develop RIAs for physical things, constrain characteristic of things must be taken into account. Due to the limit of computation and communication resources in things, they cannot afford to provide RIAs like conventional Application Servers. Instead, RIAs for physical things are contributed by separating static and dynamic web contents. Static web contents that make up rich web experiences such as HTML, JavaScript, or Multimedia are heavy so that they are offloaded to a dedicated server called Presentation Server (PS) or virtual

Presentation Instance in a cloud system. Dynamic web contents that are things' contextual information and actuation commands are served by physical things.

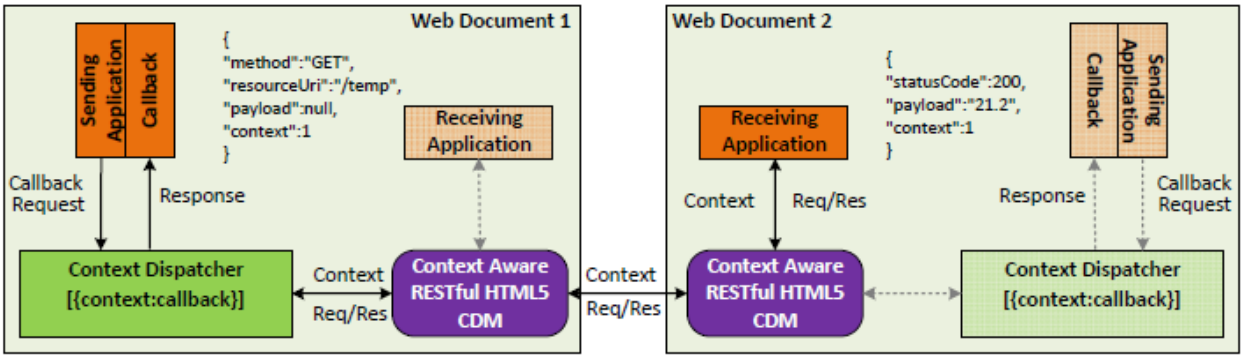
The steps how to provide RIAs to constrained things is described as follow, assuming that things are connected to the Internet and have their own domain name. In the Fig. 4-1, after resolving the thing's domain name, the client's web browser starts fetching the thing's homepage. This homepage is a tiny HTML document and is served by the thing over a lightweight HTTP/TCP stack. After that, static web contents including web templates and client scripts are downloaded from the PS to the client's web browser. Lastly, those client scripts interact with the thing to get its live data and display those data on top of the rich web template. By doing this, even the most constrained thing can be supported with RIAs directly without any middleware.

The following sections will describe all the components in detail, under three domains: client, presentation cloud, and IP-WSN.

4.2. Client domain

In client domain, end users interact with physical things through web applications and services using a web browser. End users navigate to physical things' websites and obtain services from there. The physical things return a simple and minimum web content that contains a JavaScript element called *Thingscript*. *Thingscript* is responsible for initiating and creating the whole web page with a web application template provided by PS. The web application template includes presentation materials such as images, videos, and application script codes in JavaScript, which will be called *Framescript* herein.

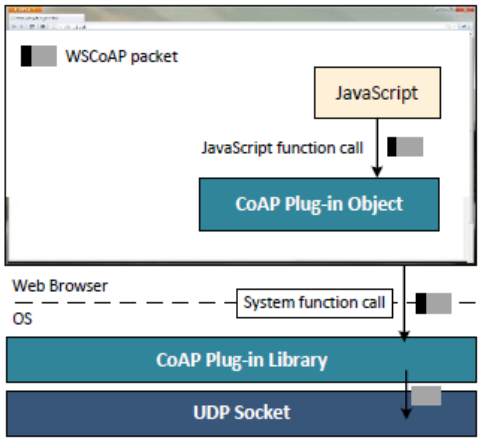
Framescript does not originated from physical things domain so that it cannot access things information. Instead, *Framescript* communicates with *Thingscript* in order to obtain things data for the application logic. This communication is carried out using an advanced implementation of HTML5 CDM called Context-aware RESTful HTML5 CDM – CRCDM. In brief, CRCDM defines a messaging standard for the communication based on RESTful web service.



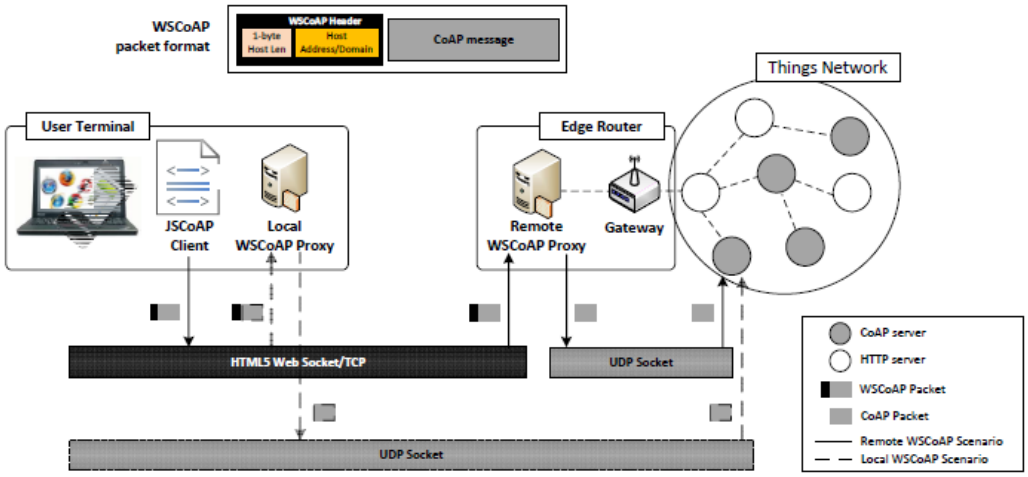
(Fig. 4-2) Context-aware RESTful HTML5 CDM

Fig. 4-2 illustrates CRCDM in detail. In this figure, there are two web documents are exchanging information with each other. They can be a normal web page and its child iframe or two iframes in one page. These two web documents are originated from different domains so that they cannot access other's information directly. In the web document 1, the sending application wants to send a HTML5 CDM message to the receiving application of the web document 2. As seen in the figure, the proposed standardized HTML5 CDM messages inherit much from the Ajax message model. Whenever the dispatcher receives a request and a corresponding call-back from an application, it generates a unique context ID and stores it along with the call-back. The dispatcher then attaches the context ID in the HTML5 CDM message and sends it out. The receiving application maintains that context ID and sends it back with the response. Using this context ID, the sender's context dispatcher can dispatch the response to the appropriate call-back of the sending application.

Apart from CRCDM, which facilitates communication between *Thingscript* and *Framescript*, communication between web browser and physical things is carried out using either HTTP protocol using Ajax technology or CoAP protocol using CoAP proxy or plugin. While the communication using Ajax technology is straightforward, CoAP protocol is a bit more challenge since CoAP is based on UDP transportation that is not supported by general web browsers. This can be overcome by using a CoAP proxy that translates HTTP to CoAP and vice versa or a CoAP plugin that attaches a UDP connection to web browsers.



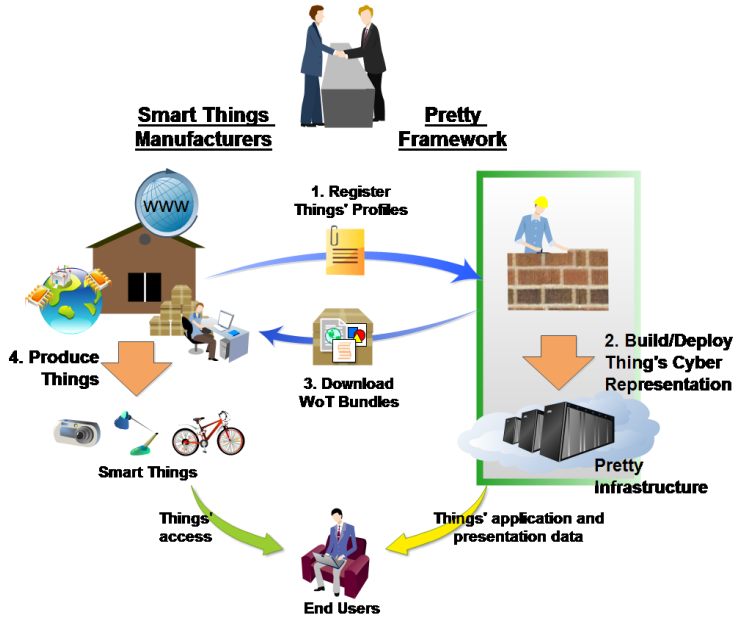
(Fig. 4-3) CoAP Plug-in Solution



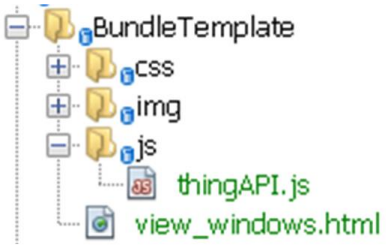
(Fig. 4-4) WSCoAP Proxy working with JSCoAP client in communicating with things via CoAP

Fig. 4-3 shows CoAP plugin solution in enabling CoAP for web browsers and Fig. 4-4 shows an advanced CoAP proxy called WSCoAP proxy. CoAP plugin is straightforward since it uses the plugin framework, which is supported by most web browsers. Briefly, in CoAP plugin the JavaScript code calls a system function, which resides in the operating system to instruct the plugin to send out the CoAP packet directly to CoAP servers over UDP. In case of WSCoAP proxy, the JavaScript code uses HTML5 WS protocol to send the CoAP message to the WSCoAP proxy over HTML5 WS packets and the WSCoAP proxy forward the CoAP message to CoAP server over UDP.

4.3. Presentation server cloud domain



(Fig. 4-5) Pretty workflow

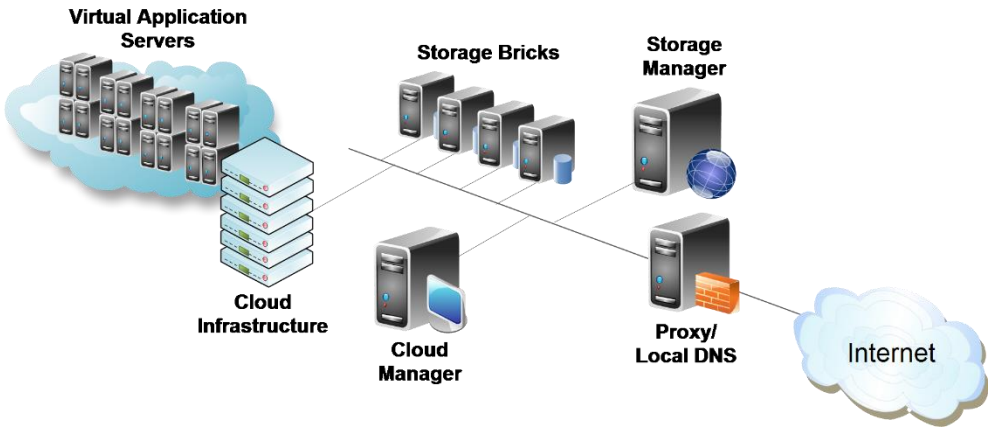


(Fig. 4-6) A Pretty WoT A/S Bundle

Fig. 4-5 illustrates the workflow in Pretty framework in creating and deploying web application for physical things. First things producers register themselves and their thing profiles to Pretty framework. In return, Pretty framework builds corresponding WoT A/S Template Bundles for the registered products and returns the Bundle to thing producers. The Template Bundle as shown in Fig. 3.9 includes a web template that is used as the thing's web interface and a JavaScript API call Thing API that is used to interact with the thing. At the same time, Pretty assigns a PS instance for storing the registered products' presentation and application data. This assignment is made depending on the system's workload. Instantiation of new virtual instance can happen in Pretty's cloud whenever more resources are required. Thing producers use the provided WoT A/S Template Bundles to develop their products' WoT A/S. The Bundle's Thing API is designed to be extremely easy to use so that WoT A/S developers do not have to worry about the system level protocols.

In this API, physical things are abstracted into a RESTful model where things' functionalities and capabilities are represented as a web resource which can be

manipulated with web methods such as GET, PUT, POST and DELETE. The developers only have to use the straightforward function calls such as "getTemperature" or "setLight('on')" to interact with physical things. After developing the WoT A/S, thing producers can upload the WoT A/S Bundle back to Pretty so that Pretty can deploy such Bundle to the pre-assigned PS instance. At this time, thing producers can proceed to manufacture their products accordingly. Later on, whenever an end user interacts with a WoT A/S enabled thing on a web browser, the thing and its cyber representation in Pretty framework are automatically linked together to provide the thing's RIAs to end users.

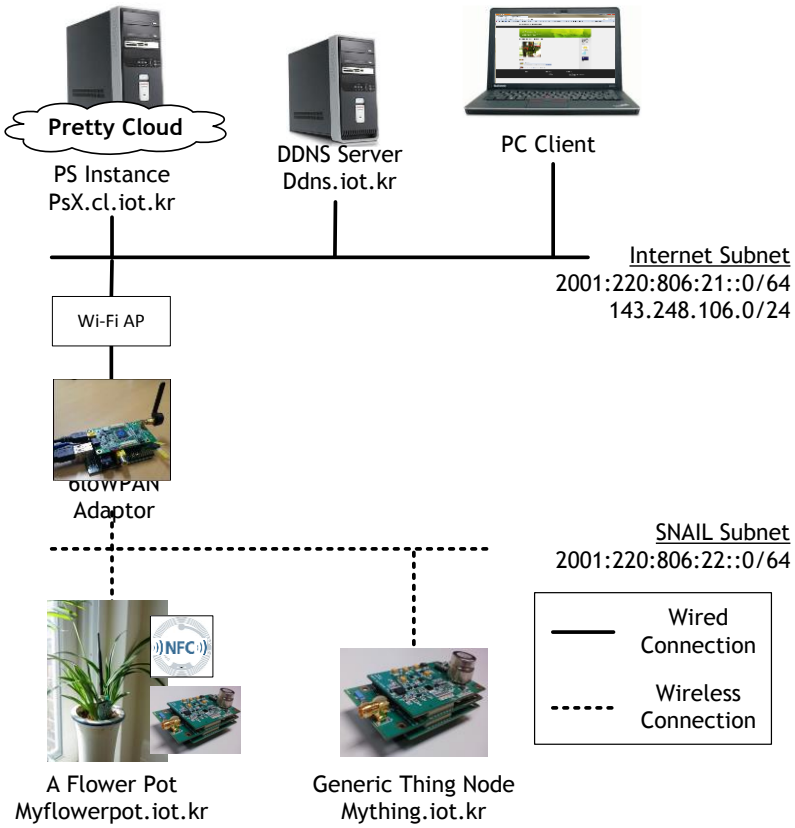


(Fig. 4-7) Pretty Application and Presentation Framework cloud system

In order to follow such workflow, Pretty uses a cloud-based system as seen in the Fig. 4-7 to generate WoT A/S Template Bundles and to provide a running environment for developed WoT A/S. On top of the cloud platform, Pretty runs its own service dashboard, networking system and database. Beside, distributed file system is used to provide a scalable and replicated shared/persistent storage for PS virtual instances.

4.3. IP-WSN domain

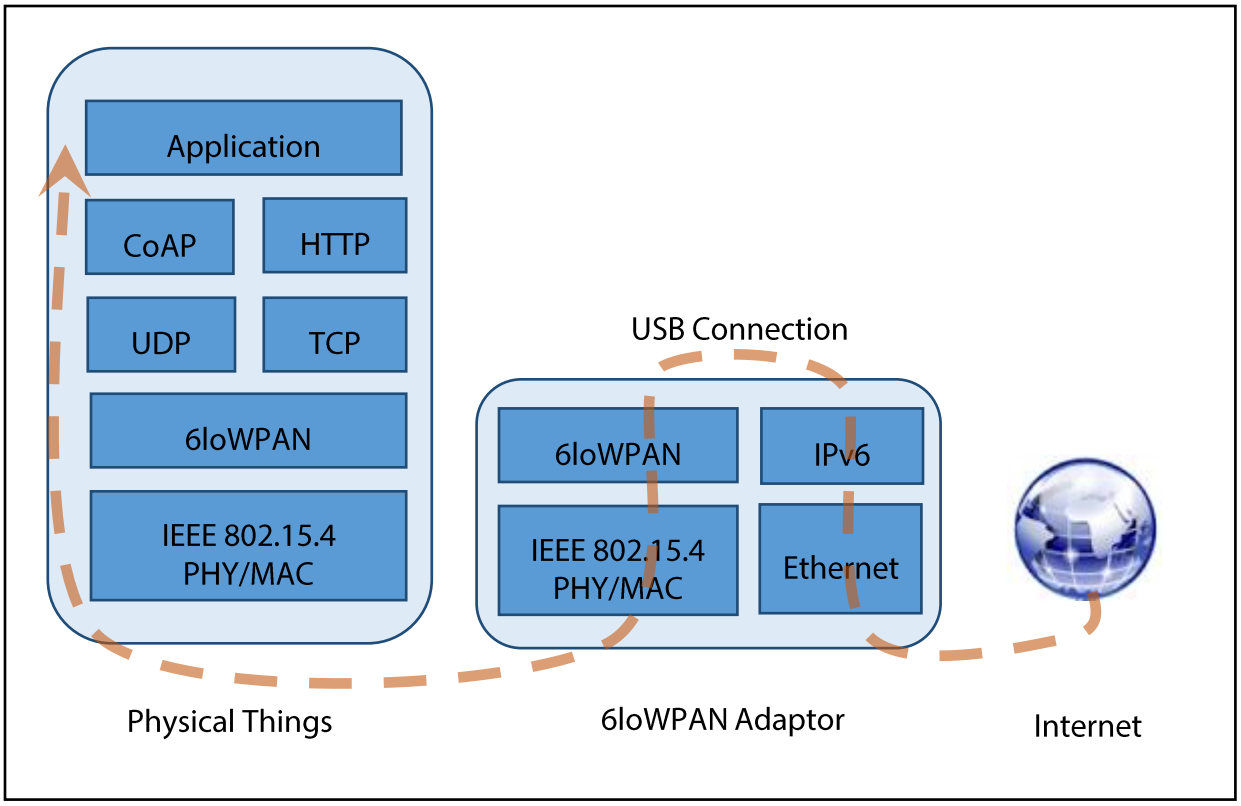
The proposed framework uses actuators and sensors in IP-WSN network as physical things. In particular, physical things is assumed to run on 6loWPAN network as described in RFC 6282



(Fig. 4-8) 6loWPAN Testbed for Pretty framework

The testbed is illustrated in Fig. 4-8. The testbed runs on two networks, an Internet subnet, and a 6loWPAN subnet. The very first and crucial elements of the testbed are 6loWPAN Sensors/Actuators, which represent smart things and 6loWPAN Adaptor, which enables web communication between web browsers and things.

To enable web technology to sensors, lightweight HTTP server and CoAP server are located in application layer, which run simultaneously on top of lightweight TCP and UDP stack. 6loWPAN network stack, which does protocol adaptation, is presented between IEEE 802.15.4 PHY/MAC layer and Transportation layer. Therefore, that, web-enabled packets can transmit on top of the IEEE 802.15.4 network. 6loWPAN Network and Application stacks are presented in detail in Fig. 4-9.



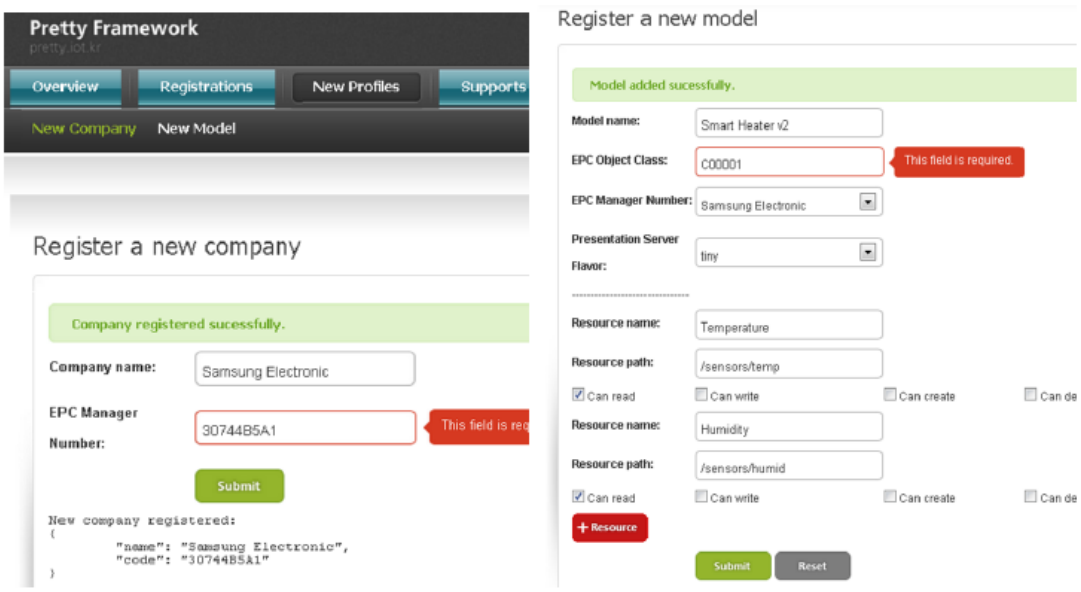
(Fig. 4-9) 6loWPAN Network and Application Stacks

6loWPAN Adaptor runs 6loWPAN software stack and is able to plug in any generic Wi-Fi access point to deploy a new IP-WSN network. Therefore, it does not require a re-design of existing network infrastructure. 6loWPAN Adaptor uses a USB port to communicate with 6loWPAN PAN Coordinator. The adaptor's kernel routing table is modified to forward packets between the Internet and the IEEE 802.15.4 network via this USB interface. 6loWPAN Adaptor also supports IPv6 auto-configuration for things so that things' IPv6 addresses are assigned dynamically. 6loWPAN PAN Coordinator is representative for a PAN; it initiates the formation of an IEEE 802.15.4/6LoWPAN network so that other things can join in.

5. Service Scenarios

5.1. Registration

It is assumed that devices manufacturers have made contracts with Pretty operator regarding price information and term of service so that user accounts are created. After logging in to the system, the user is brought to the Pretty Dashboard where he/she can manage his/her subscription. Fig. 5-1a and 5-1b show the registration process, which allow users to register their companies and corresponding models. Register a new company should be straightforward since only a company name and its EPC's Manager Number are required. However, the system will check for the uniqueness of the given Manager Number to prevent input mistake. Register for a new model requires more information since it will be used to build the appropriate SDK to develop the model's WoT A/S. This step assumes that the registering models can afford HTTP or CoAP protocols in order to interact with it from the Internet.



(a) Register a New Company (b) Register a New Model

(Fig. 5-1) Company and Model Registration

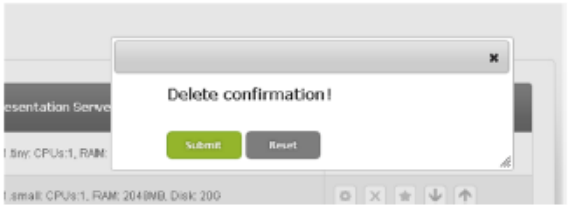
5.2. Service management

(a) Companies listing

(b) Models listing

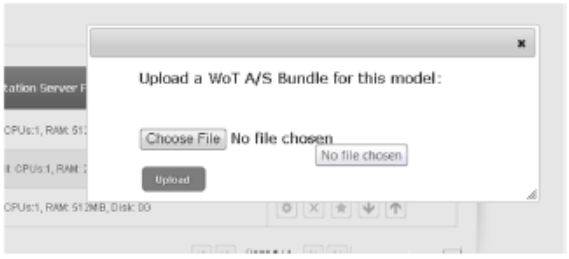
(Fig. 5-2) Company and Model Management

After registering models and companies, users can return to management pages, which are company listing and models listing as shown in Fig. 5-2a and 5-2b respectively. In these pages, users can navigate through all the registered companies and models to view, update, or delete elements.



(a) Delete a Model

(b) Model's Usage



(c) Upload a WoT A/S Bundle

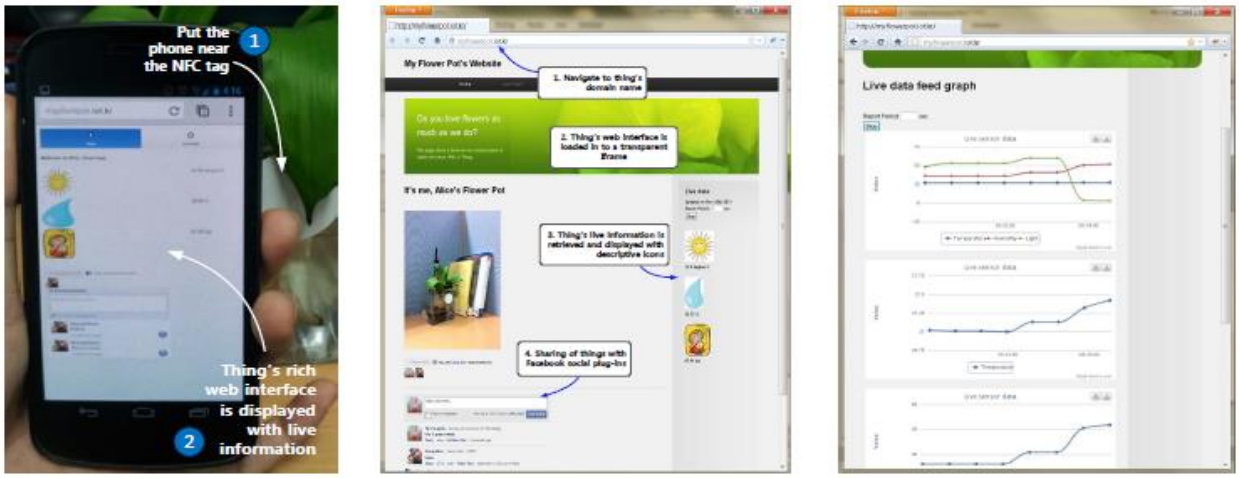
(d) A WoT A/S Bundle Template

(Fig. 5-3) Model Management

Fig. 5-3 presents some use cases that manipulate over models such as delete a model, and its usage, preview the model's WoT A/S or uploading WoT A/S Bundle for that model as well as downloading its WoT A/S template. First, developers download the model's WoT A/S Bundle template as a zipped file and extract it to develop their WoT A/S. As shown in Fig. 5-3d, this bundle includes a web template and Thing API, which are used to develop WoT A/S for the model. Next, developers create their own WoT A/S using the bundle template; make sure that not all the required library inclusions are altered. Then after finishing the development, they wrap everything back to a zipped file and upload it to Pretty. From this moment, Pretty will extract the bundle, deploy it to the designated PS instance, and make it ready for use. Finally, developers have to go back to model management page and click on the "Usage" as shown in the Fig. 5-3b. This page gives a HTML snippet that is supposed to be used as things' homepage. Thus, the developers make this snippet as the things' homepage and then they are ready for production. Whenever a user navigates to a thing's homepage, this HTML snippet is downloaded to his/her web browser and the whole WoT A/S of such thing is loaded.

Annex : Demonstration

The demonstration scenario is about Alice with her owner pot and is shown in Fig. 6-1. In our IoT testbed, Alice can navigate to her flower pot's web site either by opening a desktop web browser and type in the pot's domain name or just by putting her NFC-enabled smartphone near the NFC tag on the pot. In this website, Alice can see a nice web interface with a Fig. of her flowerpot and the pot's live status such as temperature and humidity. An emotional symbol (e.g., a smiling face) can be displayed to reflect the pot's condition.



(a) WoT A/S in Smartphone (b) WoT A/S in Desktop (c) Sensor graph

(Fig. 6-1) Model Management

Furthermore, since the pot's web interface is an iframe, which points to a web template in the PS, there is unlimited possibility for user interfaces and applications that can be obtained. For instance, Fig. 6-1b shows how social plug-ins such as the Facebook's \Comments box" and \Likes button" can be easily attached onto the iframe to share the flowerpot among Alice's friends. Another example is the web interface can be intelligently adapted to smartphone web browsers as seen in Fig. 6-1a instead of just miniaturize the desktop version. Yet another web application is demonstrated in Fig. 6-1c, which can display live graphs of environment sensing information over time and store these graphs in the web browsers' local storage. These demonstrations have clearly shown how rich web browsing experiences for resource-constrained things can be achieved with the proposed architecture.