2013 IEEE 10th International Conference on Services Computing

IoT Mashup as a Service: Cloud-based Mashup Service for the Internet of Things

Janggwan Im, Seonghoon Kim, Daeyoung Kim
Department of Computer Science
KAIST
Daejeon, South Korea
{limg00n, shkim08, kimd}@kaist.ac.kr

***Abstract*—Mashup, a way to compose a new service from existing services, is expected to play a great role in internet of things (IoT) environment. Many recent researches suggest that mashup in IoT environment is possible with existing web mashup technology if each thing exposes its functionalities as a web service. However, this approach may have limitations in dealing with many heterogeneous devices and computation scalability in the presence of large number of things involved. In this paper, we propose a cloud-based IoT mashup service model, called IoT Mashup as a Service (IoTMaaS), to overcome heterogeneity of devices by following the model driven architecture principles and computational scalability based on cloud computing paradigm. We also design a cloud platform on which IoTMaaS be executed in harmony with stakeholders such as end users, device manufacturers, and cloud computing providers. We proved the concept of the architecture by implementing a prototype platform with an vacant room detection application.**

***Index Terms*—internet of things; service modeling; cloud computing; model-driven architecture**

I. Introduction

Internet of things (IoT) [1], [2] environment provides the human with many useful services by connecting devices to the Internet. We can monitor interested places using IP cameras anywhere and anytime. We may also want complex use cases such as motion detection, face recognition and alerting to the users or police depending on the situation. Mashup, a way to compose a new service from existing services, is expected to play an important role in these cases because people have different use cases and preferences; a useful service to someone may be unnecessary or even inconvenient to others. In addition, mashup technology gives a benefit of low cost service deployment.

Many recent researches suggest that mashup in IoT environment is possible with existing web mashup technology if each thing exposes its functionalities as a web service. [8]–[12] However, mashup in IoT environment has more challenges because of large number of devices, and their heterogeneity and availability. While a web service is provided in well-known domain name and available almost all the time in web environment, web services from devices in IoT environment are heterogeneous depending on the functions of the devices. They may not even be available all the time.

In addition, amount of computation resource necessary for mashup services changes according to data which is produced by things and processing algorithm. For example, manufacturers have use case to analyse the usage pattern of the washing machine. Due to large number of devices, amount of data is large, and accordingly necessary computation power became large. Computation resource adaptation will be more significant if mashup services have timing constraints for processing real-time streaming data. Devices are expected to produce real-time streaming data, so this issue become more critical. Even if the data may be processed by external web service, mashup service is still responsible for the performance requirement. Therefore, applying current web mashup technology may not be possible in IoT environment. Instead, a new mashup service model needs to be designed which overcomes aforementioned challenges.

We propose a new mashup service model, called IoT mashup as a service (IoTMaaS), defined as composition of thing model, software model, and computation resource model. During mashup process, three components can be customized depending on end users' preference; end users can select things, processing software, and amount of computation resource to use at run-time. This model relieves heterogeneity of devices following the idea of model driven architecture (MDA), bringing interoperability among devices and mashup service and not restricting them on the specific platform and protocol.

We also propose a cloud platform, called IoT Service Cloud, on which IoTMaaS is served. Similar to IoTMaaS model which consists of three components, IoT Service Cloud can also be divided into three parts: thing configuration, software component assembly, and allocation of cloud resource. This architecture is also designed to be an ecosystem reflecting business roles of the real-world stakeholders such as device manufacturer, device owners, cloud computing providers, software developers, and end users. In other words, each stakeholder has an incentive to use the new service model. We proved the concept of our proposed architecture by implementing a prototype architecture and vacancy room detection scenario.

The remainder of the paper is organized as follows. In section 2, related works are discussed. IoTMaaS is designed in section 3. A reference architecture to serve IoTMaaS is described in section 4. The feasibility of IoT Service Cloud is proved in section 5 via the prototype implementation. Finally,

978-0-7695-5026-8/13 $26.00 © 2013 IEEE
DOI 10.1109/SCC.2013.68

we conclude our work and suggest future works in section 6.

II. Related Works

Physical mashup [9], [10] suggests that the service composition in IoT environment is possible by existing web based mashup technology if things expose its functionality in a lightweight RESTful web service. One directional characteristic of web service is overcome by using the real-time web technologies such as Comet and HTML5 Websockets. Web service, however, was designed for web documents, not for physical devices, so physical mashup uses resource description framework-in-attributes (RDFa) and microformat which are markups designed for describing the semantic of the web documents.

Following these approaches, the concept of IoTMaaS can be composed anyway using the web service mashup technologies if processing algorithm is provided as a form of open API. But, in this case, end user requirements such as real-time constraints are difficult to meet because concrete implementation of open API is hidden behind the web service interface. In addition, mashup service should be able to switch the data flow from things to open API without passing through mashup service for the efficient processing. In other words, whether letting open API call devices' web interface or letting devices call open API should be possible. However, this is not possible in the web mashup approach without the additional protocol support.

In semantic enhanced service proxy approach [11], the functionalities of Things are exposed as web service. Exposed web service can be described in domain ontology based service description as well as WSDL based service description, and micro-formats description. They call this concept sensor-as-a-service. In short, semantic enhanced service proxy approach exposes sensors as web service. This approach is similar to physical mashup approach except that this uses semantically enhanced web service description, so it has similar problems to physical mashup approach.

SOCRADES [12] also exposes smart devices with embedded software as a form of service using DPWS which is a subset of standard web service interface. The exposed services are composed according to the description in extended BPEL and integrated with enterprise systems such as ERP. This approach has similar to physical mashup except that this exposes thing service using DPWS. It also has similar problems to physical mashup approach.

Atlas [18] is early stage IoT platform. It consists of sensor node, hardware or software platform, and service gateway framework. All the Things are abstracted in the unit of node in Atlas platform. Middleware is implemented in OSGi so that dynamic service composition and discovery is possible. However, middleware in Atlas is assumed to be working in the centralized server in a place like home. Scalability will be a critical issue when the number of things and data to process increases.

In addition to three types of cloud computing services, we propose another type in this paper. Cloud computing services

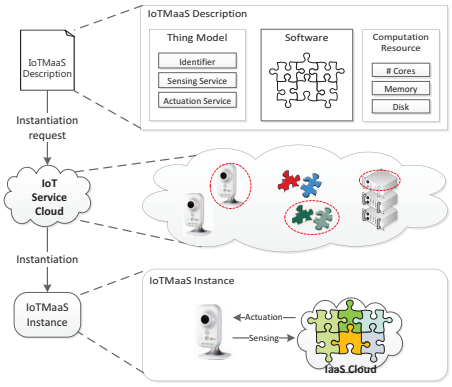


Fig. 1. IoTMaaS Concept

can be categorized into three types; software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). SaaS means that clients are served with complete software like Gmail. PaaS means that clients are served with a platform on which they can develop their applications. IaaS means that clients are served with virtual machines on which they can deploy their software. IoTMaaS is suggested as a new category of cloud services. It provides end users with an interface to customize a mashup service at runtime. Their customization options include things, software, and computing resource. The mashup service communicates with things to collect and process data on an allocated computing resource.

Mashup-container in Platform as a service (PaaS) environment [14] proposes an architecture for event-driven mashup and deployment. Mashup map in XML format is deployed in Service Execution Platform (SEP) which consists of orchestrator and service proxy. External services are wrapped in a service proxy, and events from external services are routed by orchestrator. If the functionality of things are exposed via web service, things can be composed with this approach. However, controlling the performance of mashup service is still difficult because mashup service does not know how the processing algorithm which is running as an external service is implemented. The way to compose physical things is not addressed in this paper, but in our paper, we suggest very concrete way to compose physical things including identification system, metadata sharing system, information maintained by thing manufacturers and etc.

III. IoT Mashup as a Service Model

A. Platform Independent Model

IoTMaaS is a new class of cloud-based IoT mashup service model. According to the service oriented architecture (SOA) principles, a service model and its realizing infrastructure are suggested independently. IoTMaaS is defined as a mashup of things, software, and computation resource as described in Fig 1. We assume that mashup service processes data

from things to produce output upon a dynamically allocated computing resource. Results may be used for notifications to users or controlling things as actuators. We can expect that most services in IoT environment can be described using IoTMaaS model.

A definition of a thing in IoTMaaS is an identifiable object which can have sensing and actuation services. When manufacturers make a product, they select an appropriate combination of input and output. While a product may have many internal sensors and actuators for dealing with input and output, respectively, manufacturers have to decide the product's functionalities to expose. These exposed functionalities become sensing and actuation service, which form a thing model with an identifier for identification of the thing in a network.

Describing the thing model in platform independent model (PIM) brings an advantage over doing in platform specific model (PSM). Describing the thing model in PSM results in restrictions on the thing implementation. For example, if thing model is described in web service, then all the manufacturers should implement their product to support web service. However, PSM can differ due to device heterogeneity and technology evolution. Some resource constrained devices may not support web services, and web service may alter to another technology in the future. Thing model in PIM can provide interoperability in these cases with a little adaptation effort. As long as PIM is the same, the things are interoperable because interoperability issue is reduced to the interface adaptation issue, which can be achieved by middleware level protocol translator or software bus bridge. Otherwise, device-level adaptation is needed. In addition, thing model in PIM also plays a role of a basic consensus about things for searching their metadata with an appropriate identification system. As a thing model in IoTMaaS, we use super distributed object (SDO) which consists of an identifier, device profile, services it is providing, relationship with other SDOs, and configuration methods.

A definition of software in IoTMaaS is an assembly description of software components. This definition came from component-based software engineering where a unit of encapsulated functions called software component (SWC) is assembled together to a complete software. There are many SWC models in the world, and they have different component models and assembly descriptions. SWC model described in PIM bring advantage as in the thing model. Following this idea, each SWC platform takes assembly description in PIM as an input, and translate it to PSM, then assemble the software according to the translated assembly description.

Thing and software model are separated in IoTMaaS even though they can be considered as the same in SWC model. In the context of mashup service, software model plays a role of processing instructions while things play a role of data sources and consumers. We also assume that things are already active state but software components become active after the assembly upon allocated computation resource. Different information is needed for thing and software configuration in mashup service.

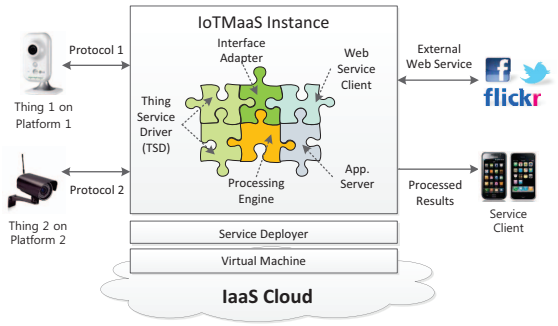


Fig. 2. An example of instantiated IoTMaaS

A definition of computation resource in IoTMaaS is a current computer model consisting of CPU, memory, disk, persistent storage, network, etc. A good example of this is Amazon elastic compute cloud (EC2). Amazon EC2 defined logical computer model which consists of ECU (EC2 computing unit), memory, instance storage, I/O capability, etc. They provide infrastructure as a service (IaaS) by translating abstracted computer model to real implementations. This should be the standard of IaaS cloud computing. So we decided to use EC2 computation model as computation resource model.

Specification of IoTMaaS includes identifiers of things which are involved in mashup service, mashup software structure which is defined as assembly description of SWCs, and amount of computation resource of mashup service. Furthermore, more dynamic specifications are possible if the platform supports. For example, we can specify attributes of things to find instead of thing identifiers and non-functional metrics that mashup service should achieve such as timing constraints achieved through computation resource adaptation depending on the processing loads.

Fully instantiated IoTMaaS works as described in Fig. 2. Things can operate on different platforms and communicate with different protocols. Regardless of these heterogeneities, SWC model in IoTMaaS treats them as a uniform model. This is possible with thing service driver (TSD) component which is a representation of things in SWC model to provide the programming interface to other software of IoTMaaS. When the interface is invoked, marshaling and unmarshaling process happens as in usual distributed objects middleware. TSD is assumed to be provided by thing's manufacturer. Other components than TSD play a role of processing engine encapsulating extensive computation algorithm, application server encapsulating communication with service clients, web service client which uses other web service, and interface adapter adapting one interfaces to another. For example, a face detection SWC requests to the TSDs to capture the frame. TSD retrieves the frame from the IP camera via proprietary protocol and returns to the face detection SWC. If face is detected, then the information is compared to the face via external web

service. The result is notified to users via application server SWC.

Interfaces of SWCs and TSDs can be known in advance, so we think of processing software structure, called an IoTMaaS template, which is later customized by end users via thing description to use, customization parameters, and computation resource allocation plan. That is, end users can customize things and computation resource at run-time.

This SWC model is executed on IaaS virtual machine instance, which was pre-configured image to have a SWC model. A software component model running on VM looks up and assembles necessary software components according to the IoTMaaS template and customization parameters.

B. Platform Specific Model

Previously mentioned PIM should be realized in the specific platform. We decided to use SDO as a thing model. SDO PIM defined in unified modeling language (UML) should be implemented onto the specific platform. The selection of the platform is dependent on the manufacturer's choice. Most distributed object middleware provides its own interface description language (IDL) and a compiler to generate client and server side code. The generated client side code will be used as a remote delegate object or TSD in the mashup service. The generated server side code will be a skeleton for manufacturers to implement the functions of things. After development of things, manufacturers should provide TSDs by wrapping generated client side code to be used in IoTMaaS SWC model.

Electronic product code (EPC) [19], a universal identifier defined by GS1/EPCglobal to uniquely identify physical things in the world, is used as an identifier of a thing. Use of EPC is advantageous for looking up thing's metadata which is usually maintained by its manufacturer. EPC has a hierarchical structure which consists of manufacturer number, item class number, and serial number, so it is possible to build a global scale metadata sharing system using DNS if EPC-derived hostnames are used for DNS lookup.

An IP camera can be represented in SDO model. An unique identifier of a thing, EPC urn:epc:id:sgtin:0614141.112345.3, is represented in the attribute sdoID. A type of thing, ipcamera, is represented in the attribute sdoType. The internal states are represented as name-value pair in the attribute Status. Physical attribute of thing is represented in Device Profile, which includes deviceType, manufacturer, model, version, etc. Services provided by a thing are represented in the list of Service Profile, which includes serviceID, interfaceType, etc. For example, IP camera can have internal state heading direction in the attribute Status. Its manufacturer like LG, model LW130W, version 1.0 can be represented in the Device Profile. Its services like capturing the frame via camera and sound via microphone can be represented in the Service Profile. Depending on the thing services, additional software other than SDO may be needed which is usually maintained by its manufacturer. So sharing this kind of software can be done via EPC-derived metadata sharing system. Note that Service Profile is not a part of a SDO because it also has unique identifier and not one-to-one relation. It means that each thing provides services which represented by its Service Profiles accidentally, in other words, by manufacturer's decision. This implies the existence of service layer above thing abstraction layer.

As for SWC model in IoTMaaS, we chose to use common object request broker architecture component model (CORBA Component Model, CCM). CCM is a SWC model which is defined based on CORBA distributed objects middleware. The reason to choose CCM is that CORBA is standardized and operating systems and programming language neutral. SWC models such as open service gateway initiative (OSGi) is Java-based, so it restricts SWC model in IoTMaaS to Java.

Computation resource described in PIM should be realized in specific IaaS implementation. Depending on the technologies, computation resource is differently realized. For example, even though Amazon EC2 defines the computation resource in ECU, slight performance difference can exist between virtualization technologies such as Xen and KVM.

IV. An architecture to serve IoTMaaS

We designed a platform to serve IoTMaaS as described in Fig 3. IoTMaaS Service Planner (SP), a professional service designer, defines IoTMaaS Template which is a SWC assembly description customized later depending on the user's request. SP is responsible for the correct and reliable operation of IoTMaaS. IoTMaaS Instantiation Client (IC), a non-professional end user client, sends instantiation request to IoTMaaS Frontend Service (FS). FS is a frontend which not only manages resources of the whole platform but also executes IoTMaaS in response to the instantiation request. FS sends a launch request to IaaS provider to execute a virtual machine on which IoTMaaS service deployer (SD) is running. FS also sends a composition request to SD, which downloads SWCs and TSDs from SWC repository service and thing profile service (TPS), respectively. Thing identifier service (TIDS) provides lookup mechanism for retrieving addresses of TPS. TPS is operated by thing manufacturers and should be registered to TIDS. SD also retrieves addresses of things for thing access via TIDS and domain name service (DNS). Finally, SD assembles the SWCs and TSDs according to IoTMaaS specification. Assembled service is called IoT Service Instance (SI), which collects and processes the data from things to notify the Service Client (SC), or to actuate things. In addition, SI can provide SC with useful user interface (UI) and user experience(UX). Thing discovery service (TDS) is used for FS to discover things and check their availability. TDS maintains a list of thing cluster service (TCS), which is used for discovery and availability check. TCS is provided by thing cluster controller (TCC), which not only manages things in the hierarchical manner but also provides access control to things. Each thing is abstracted to thing model via thing controller (TC), which exposes thing's functionalities as a form of thing service.

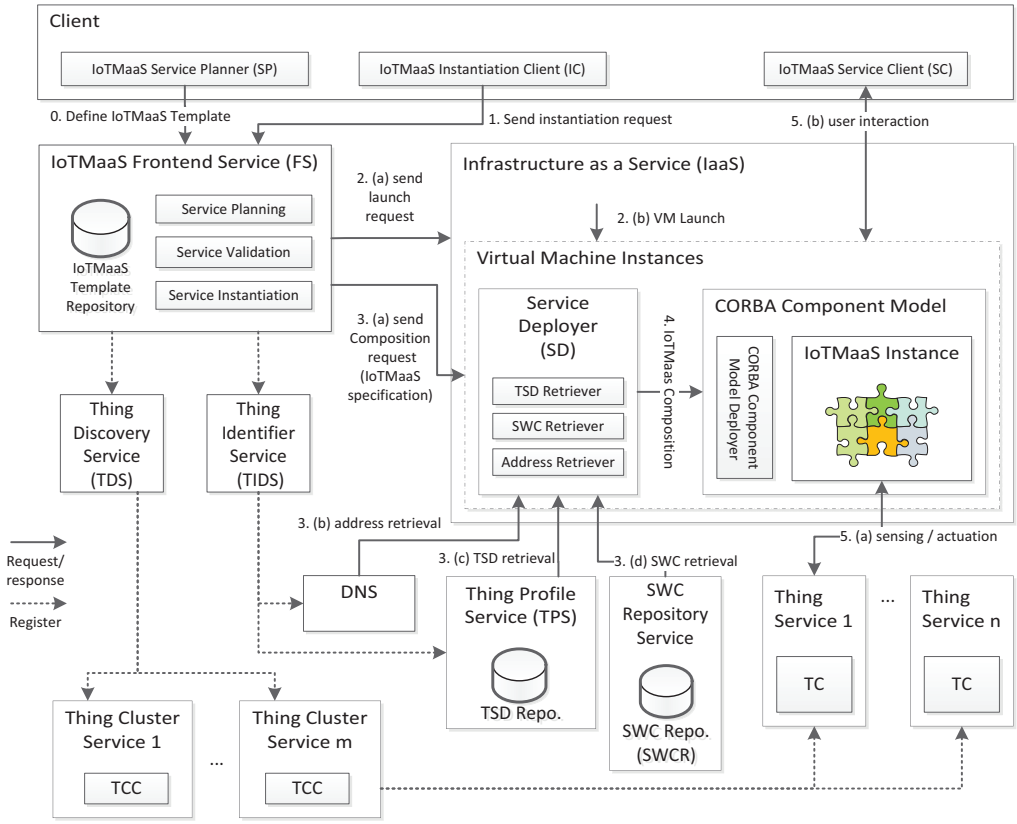


Fig. 3. An architecture to serve IoTMaaS

A. Thing Cluster and Discovery Service

To mash up IoTMaaS, things and their management infrastructure should exist. There are several ways to manage things: centralized, hierarchical, and distributed way. Centralized way is that all the things in the world should register themselves to the centralized registry which provides monitoring and management functions, but this is difficult to realize because thing management role is inherently distributed among thing owners. Distributed way is that things maintain the monitoring and management infrastructure in peer-to-peer, but large amount of network traffic should be exchanged among things. This may be not supported by resource limited things. In our design, things are registered into a cluster which is a hierarchically structured group of things used for monitoring, and management functions.

Basically, the role of a thing cluster is to register things and maintain registry information which will be used for searching appropriate things upon discovery request. Registration process can be initiated either by things or TCC. After registration, TCC uses lease approach to measure the availability and connectivity to things. In other words, things should periodically report their status to TCC. Otherwise, they are considered disconnected and unavailable.

TCCs are registered to Thing discovery service (TDS), which is used to find out the appropriate things upon FS's discovery request. It returns thing's identifiers matched with the user's interest to discover. The identifiers are later used for IoTMaaS instantiation request.

Thing controller (TC) abstracts each thing into thing model. Regardless of the protocols and platforms they use, things can be treated as uniform model through TSD in SWC model. TC monitors the resource of each thing and maintains its availability information. TC also reports the current status information to TCC periodically to renew the lease time which is used to check thing availability.

B. Thing Metadata Retrieval and Service Composition

While thing cluster controller (TCC) and thing discovery service (TDS) manages dynamic information about things, static information maintained by the manufacturers such as product information and thing service driver (TSD) should also be shared. This information is shared based on thing identifier service (TIDS). TIDS transforms identifiers of things to domain name for address retrieval and addresses to the thing profile service (TPS) for retrieval of inherent metadata and

TSDs. Interface of TSDs is known when IoTMaaS template is defined, but implementation of TSD can be different from things to things so that thing manufacturers are able to use different middleware bus for communication between things and TSDs. Nevertheless, TSDs abstracts heterogeneous things to uniform thing model.

SWCs other than TSDs are downloaded from the URLs which are embedded in the IoTMaaS template. After all the TSDs and SWCs are downloaded, they are assembled according to the assembly description in the IoTMaaS template. End users can customize the SWCs by specifying parameters of SWCs. For example, end users can adjust the threshold value of the motion detection software.

Although use of SWC model brings adaptability and reconfigurability, SWC assembly is too difficult to handle for end users. So, in the architecture, we designed a role of service planner (SP) to define IoTMaaS template. SP is responsible for ensuring reliability and availability of IoTMaaS.

C. Steps to compose IoT Service

Even though IoTMaaS is defined in terms of things, software and computing resource, the architecture does not compose it at once. Instead, we distribute composition roles among the subcomponents of IoT Service Cloud. The composition steps are divided into five stages.

First, IoT service instantiation client (IC), a nonprofessional end user, decides the IoTMaaS template, customization parameters, and thing identifiers. IoTMaaS template should be defined by IoTMaaS Service Planner (SP) previously. IC selects a service template among the IoTMaaS template list provided by FS. IC also can customize IoTMaaS by specifying not only parameters of SWCs but also thing identifiers. Identifiers can be discovered through thing discovery service (TDS), and IC selects among them. In addition, IC specifies the amount of computing resources such as number of CPU cores and memory in the request. IC sends the instantiation request to FS with these information.

Second, After validation process, FS launches the preconfigured virtual machines according to the instantiation request. In the validation process, FS checks availability of things, TSDs, and SWCs. Then, FS sends launch requests to IaaS provider such as Amazon EC2. Virtual machine image should be preconfigured to run SD after booting.

Third, FS sends the composition request to SD, and SD prepares SWCs, and TSDs according to the composition request. Composition request consists of SWC assembly description, customization parameters and URLs which is used for TSD and SWC retrieval. According to the composition request, SD retrieves SWCs from the software component repository (SWCR). It also lookups TPS of things via TIDS and retrieves TSDs from the TPS. Thing address is also retrieved via TIDS and DNS.

Fourth, SD composes IoTMaaS Instance according to the assembly description in the composition request. SD generates archive files which include assembly description, TSDs, SWCs, thing address, and customization parameters. Then, SD sends the deployment command to the CORBA component model (CCM) deployer with the archive files.

Fifth, IoTMaaS instance begins to provide the service by connecting to things and getting the data from them. SWCs runs the service logic which processes the data to get the result. Depending on the result, it can notify to SC with useful UI/UX or actuate the things.

D. Computing Resource Allocation

Computing resource means the specification of virtual machine (VM) resources to run IoTMaaS. The specification of computing resource includes the number of virtual machines, the number of cores, the memory and the storage of each VM. To support more IaaS providers, it is advantageous for the platform to be compatible with cloud computing standard. EC2 interface which is the interface used by AWS EC2 is considered as de-facto standard because most IaaS providers implemented EC2 interface to make their IaaS compatible to AWS EC2. In our architecture, we chose to use EC2 interface. It brings interoperability to any EC2-compatible IaaS, making our architecture independent of IaaS providers.

In this stage, the computing resource may be allocated at run-time after consideration of the number of things used, the data generated by things, and the complexities of processing algorithm. Resource computation and adjustment depending on the computation loads is out of scope of this paper, but this feature can be added in the future because IoTMaaS is operated on the component based software platform.

E. Towards an Ecosystem

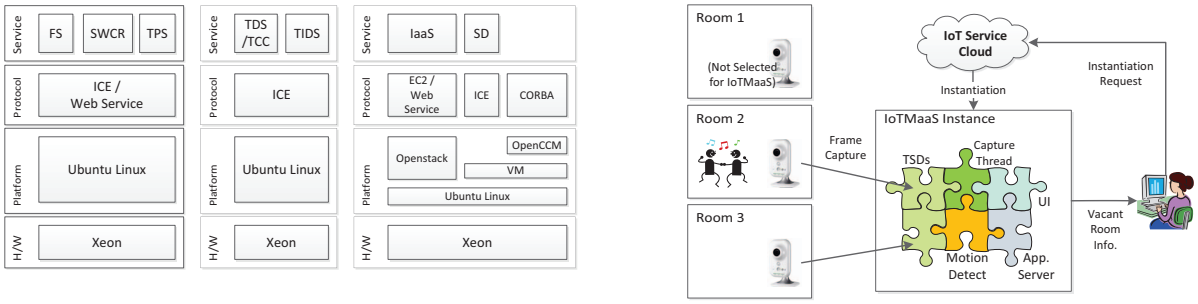
We expect this platform may work as an ecosystem. IoTMaaS Instantiation Client (IC) can charge IoTMaaS Service Client (SC) for the usage of the IoTMaaS. IoTMaaS Service Planner (SP) can also charge IC the usage of the IoTMaaS. SP have to pay for the usage of IaaS, software components (SWCs), and things to IaaS provider, third party software developers, and thing owners. In each payment stage, each role have the responsibility to make his own service work correctly and reliably. It is natural for each role to take some portion of margin. We expect this ecosystem will be realized using our platform.

V. Implementation

A. IoT Service Cloud Platform

We implemented the prototype of the platform to evaluate the feasibility. Fig. 4(a) describes the overall system.

FS is implemented in Ubuntu Linux 12.04 server on HP server (Xeon Quadcore E5504 2.00GHz, RAM PC3-10600E 4GB, Gigabit LAN, HDD 500GB). Its interface is Ajax-based web interface which is developed using Google web toolkit (GWT). User interface (UI) of FS provides configuration of VMs and IoTMaaS. SWC repository (SWCR) and TPS are implemented as a simple web server for SWC downloads. TCC and TIDS are implemented on Ubuntu Linux in another machine whose specification is the same as above HP xeon machine. They function as a server for thing information and



(a) IoT Service Cloud Platform

(b) IoTMaaS Application Scenario

Fig. 4. A Prototype Implementation

(a) Frontend Service UI

(b) Room status and Example IoT-MaaS UI

Fig. 5. A Prototype Service UI

do not provide user interface. These components communicate each other using internet communication engine (ICE) which is a compact distributed objects middleware.

SD is implemented in the Openstack VM on which Ubuntu 12.04 server is running. OpenCCM 0.9.0, an open source implementation of CORBA Component Model on Java platform, is used as SWC model. SD receives composition request via ICE, downloads components from SWCR and TPS and execute OpenCCM deployer. We used JacORB 2.2 as CORBA object request broker (ORB).

IaaS is implemented using Openstack on HP Xeon server which is the same as FS. All the components are installed in a single machine for simplicity. Devstack script, an automated installation script of Openstack, is used for installation. Openstack is accessed by FS through Euca2ools which is a command line tool for Eucalyptus and also compatible to EC2 and Openstack.

B. Vacant Room Detection Scenario

An example IoTMaaS is a vacant room detection scenario. In this scenario, IP camera is installed in each room for other purpose. End user wants to find vacant rooms using IP camera. Some IoTMaaS service planner (SP) defined a service template to detect vacant room through the motion detection algorithm. The end user chooses the service template to use and customizes the cameras which are in the interested room.

We assume that three LG W130W IP cameras are installed in three rooms. For the demonstration purpose and limitation of the devices, we organized some part of scenario manually. LG IP-camera is using proprietary platform, and the manufacturer should have provided TSDs to use, according to our platform. But the manufacturer did not provide TSDs, so we need to develop TSD by ourselves. Because LG IP-camera supports RTSP protocol, we developed a client side software of RTSP using FFMPEG library in JavaCV. Then, we wrapped it to the CORBA SWC. The customizable parameters of the TSD are identifier and address of things. This TSD is retrieved from thing profile service (TPS) after lookup through thing identifier service (TIDS). This IP camera does not support thing abstraction and does not have internal GPS. So, we abstracted it with external thing controller (TC). The abstracted thing values such as the location is manually input, so they may not the real value.

Vacancy Detection S/W is provided as a form of SWC from SWCR. It needs the capture service which is provided by TSDs. Motion detection software is developed using JavaCV. It simply compares the current frame with the previous frame. If the difference is above the threshold value, then it detects the movement. The movement is regarded as non-vacancy of the room. The software is also wrapped into the CORBA component. The customizable parameter is threshold value.

Application server component is also provided as a form of SWC from SWCR. It provides end users with web based UI to show vacant room information. The interface is implemented using Google web toolkit (GWT). End user can get the information using a simple web browser. In this scenario, IoTMaaS Instantiation Client (IC) and Service Client (SC) is

the same, the aforementioned end user.

Fig 5(a) shows the user interface of mashup UI which is running on frontend service (FS). Mashup UI is divided into two parts: VM configuration and mashup service configuration. In VM configuration part, end users can launch the VM according to the computation resource description. For the demonstration purpose, we designed mashup UI to be as simple as possible even though there are many possible customization options; Users can only launch and terminate VMs whose size is pre-assigned. After booting, VMs become the active state meaning that they are ready to receive composition request. Next, end users generate composition request by configuring mashup service part. End users select the service template which is an assembly description of software. In this UI, VacantRoomDetection template is selected. Then, end users select things to use by configuring TSDs. The type of thing services should be selected first, and things providing selected service are displayed on the map. End users select the things to use by clicking them, and the color of selected things is changed to green. Finally, end users can customize the parameters of SWCs such as motion detect threshold value. If the user click instantiate button, the composition request is sent to the service deployer which is running on the selected VM.

Fig 5(b) shows the status of the rooms and the UI for an example IoTMaaS. Two upper figures show occupied and vacant room. The bottom figure shows that two cameras are used as we selected in the FS UI. Green-colored marker means vacancy, red-colored one is occupied state. If motion is detected, the color is changed to red. We can reset the vacancy information by clicking reset button.

VI. Conclusion and Future Works

We proposed a cloud-based mashup service in IoT environment which is a composition of things, software, and computing resource. According to the end users preferences, three components can be customized. The contribution of this paper is that heterogeneity of devices is relieved by specifying the model in both PIM and PSM following the MDA principle. In order to realize the model, we also designed a cloud platform on which IoTMaaS is served. In our platform, mashup roles are divided to stakeholders, so we expect the platform can function as an ecosystem. The concept is proved by prototype implementation of the platform and a vacant room detection scenario.

For the future work, current composition model needs to be refined to accommodate use cases from diverse fields. For example, some applications may need timing constraint where computation should be completed before the deadline while number of things used is changing. In this case, computing resource should be adjusted depending on the computational loads at run-time. Elastic stream processing framework such as S4 [13] can be exploited in the future work. Other applications may need safety constraint. When things which were previously controlled by mashup service are disconnected, things should be automatically change their state to the safe state. Otherwise it may bring disasterous situation.

Acknowledgment

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (NRF 2012-0008824) and also supported by the SW Computing R&D Program of KEIT(2012,10041313, UX-oriented Mobile SW Platform) funded by the Ministry of Knowledge Economy.

References

[1] A de Saint-Exupery, Internet of Things - Strategic Research Roadmap, CERP-IoT, 15 SEPTEMBER, 2009

[2] L. Atzori, A. Iera, and G. Morabito, The Internet of Things: A Survey, Elsevier Computer Networks, 2010

[3] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner., A break in the clouds: towards a cloud definition. SIGCOMM Comput. Commun. Rev. 39, 1 (December 2008)

[4] Object Management Group (OMG), Super Distributed Objects, 1 October, 2008.

[5] J. Sung, Y. Kim, T. Kim, Y. J. Kim, D. Kim, "Internet metadata framework for plug and play wireless sensor networks," Proc. Sensors Applications Symposium (SAS'09), IEEE Press, Feb. 2009, pp. 320-324

[6] Object Management Group (OMG), Common Object Request Broker Architecture (CORBA) Specification, Version 3.1.1 Part 3: CORBA Components, 3 August, 2011.

[7] Amazon Web Service Elastic Compute Cloud (AWS EC2), http://aws.amazon.com/ec2/

[8] 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.

[9] D. Guinard. Mashing up your web-enabled home. In Adjunct Proc. of ICWE 2010 (International Conference on Web Engineering), Vienna, July 2010

[10] D. Guinard, V. Trifa, and E. Wilde. A resource oriented architecture for the web of things. In Proc. of Internet of Things 2010 International Conference (IoT 2010), Tokyo, Japan, Nov. 2010

[11] Sarfraz Alam, Josef Noll, "A Semantic Enhanced Service Proxy Framework for Internet of Things", Proceeding GREENCOM-CPSCOM '10 Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing IEEE Computer Society.

[12] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. M. 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-Volume 00, 2009, p. 968975

[13] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, S4: Distributed stream computing platform, Data Mining Workshops, International Conference on, vol. 0, pp. 170177, 2010.

[14] Michele Stecca, Massimo Maresca, "An Architecture for a Mashup Container in Virtualized Environments," cloud, pp.386-393, 2010 IEEE 3rd International Conference on Cloud Computing, 2010

[15] EPCglobal Inc., EPCglobal Architecture Framework v1.4, http://www.gs1.org/gsmp/kc/epcglobal/architecture

[16] Sergei Evdokimov, Comparison of Discovery Service Architectures for the Internet of Things, 2010 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing.

[17] Brnsted, J.; Hansen, K.M.; Ingstrup, M.; , "Service Composition Issues in Pervasive Computing," Pervasive Computing, IEEE , vol.9, no.1, pp.62-70, Jan.-March 2010

[18] King, J.; Bose, R.; Hen-I Yang; Pickles, S.; Helal, A.; (2006), "Atlas: A Service-Oriented Sensor Platform: Hardware and Middleware to Enable Programmable Pervasive Spaces," Local Computer Networks, Proceedings 2006 31st IEEE Conference on , vol., no., pp.630-638, 14-16 Nov. 2006

[19] EPCglobal Inc., "Tag Data Standard (TDS) Version 1.6", Available at http://www.gs1.org/gsmp/kc/epcglobal