2016 IEEE International Conference on Mobile Services

# Managing On-demand Sensing Resources in IoT Cloud Systems

Duc-Hung Le, Hong-Linh Truong* and Schahram Dustdar

Distributed Systems Group, TU Wien, Austria
E-mail: {d.le, truong, dustdar}@dsg.tuwien.ac.at

***Abstract*—The changes in software development and operation for Internet of Things (IoT) and the increasing integration between IoT and cloud systems have fostered on-demand sensing resources deployment for advanced applications. Such a deployment relies on various software sensors and dependent components that can be pulled from repositories and marketplaces to establish on-demand resources for sensing. To enable the utilization of such resources in a pay-per-use fashion and to reduce operational costs, in this paper, we provide dynamic up-to-date information about sensing resources at runtime to support applications to deal with dynamic changes of requirements, such as changing communication protocols between IoT devices and cloud systems, launching and activating sensors, or changing cloud services to support sensing data. Our approach is to collect and correlate both static and runtime information from different software stacks of sensing resources in the existence of the diversity, dynamism and large-scale associated with IoT cloud systems. To this end, we develop a framework for collecting, managing and provisioning information of on-demand sensing resources. We design and implement a rich set of data collectors and a middleware to gather information through different phases in the lifecycle of sensing resources. We demonstrate the extensibility, flexibility and usefulness of our framework with several examples.**

## I. Introduction

Internet of Things (IoT) cloud systems are an emerging model in which IoT elements blend with cloud services to create a unified system running on distributed infrastructures of IoT devices and cloud data centers [1]–[3]. The IoT elements (e.g. sensors, lightweight analyzers, gateways, and actuators), and cloud services (e.g. storage and data processing) in a single IoT cloud system need to be managed in a coordinated manner to ensure that applications atop such a system can seamlessly access capabilities of IoT elements and cloud services. Due to changes in virtualization and software-defined capabilities in IoT infrastructures [4], [5] and business models in provisioning IoT resources [6], [7], for sensing and analysis purposes, sensing resources – like software sensors, actuators and lightweight analyzer – are pulled from different cloud-based repositories and marketplaces, deployed into IoT devices at the edge of the network, and then controlled to collect and analyze information about "Things" (sensors and actuators can be also the "Things" themselves). This creates very dynamic virtualized resource instances for sensing purposes, while helps saving energy and management cost. However, this way also brings great challenges for managing information about these resources for sensing and analysis.

When developing applications utilizing such on-demand sensing resources, the developer needs frameworks and APIs to support her/him to access comprehensive and up-to-date information about IoT elements and the underlying systems [1]. This is because the developer has to provide features within applications to deal with the complexity and dynamism of such IoT sensing resources and their cloud counterparts. However, these types of information do exist at different phases of the IoT sensing resources life-cycle: (i) at the development phase, sensing resources are created, published, and offered by resource providers, (ii) at the deployment phase, the application developer tailors these resources, designs the applications and deploys the application and sensing resources on suitable infrastructures, and (iii) at the operation phase, the applications and infrastructures are monitored and managed by many management services, such as for deployment, elasticity control and performance monitoring. All these types of information can be used for dynamically configuring the application and their sensing resources on-the-fly.

However, capturing and managing these types of information are cumbersome. First, on-demand sensing resources are dependent on, e.g., IoT infrastructures and cloud services, so we need to consider their relationships to capture enough information. Second, the information from different sources must be correlated to ensure the consistency and to provide a unified comprehensive system view. Finally, due to the large-scale of the systems (e.g. contains thousands IoT devices and services), we also need a flexible and extensible management mechanism to sustain the changes of the systems. In this paper, we contribute a novel framework – called ELISE (Elastic Configuration Information as A Service) – to address the above-mentioned challenges by collecting and providing different types of sensing resources information. This framework provides, at runtime, comprehensive information of on-demand sensing resources in IoT cloud systems for developers, applications and other management services to decide and optimize their operations. Moreover, APIs and components of the framework can be used during the development of applications in IoT cloud systems. We setup a testbed of large-scale deployments of IoT cloud systems to show the flexibility and extensibility of our framework.

The rest of this paper is organized as follows: Section II presents motivation and related work. Section III presents our methods for collecting sensing resources information.

This work was partially supported by the European Commission in terms of the the CELAR FP7 project (FP7-ICT-2011-8 #317790) and the H2020 U-Test project. We thank Georgiana Copil, Daniel Moldovan and Tien-Dung Nguyen for their fruitful insights.

*Corresponding author

2329-6453/16 $31.00 © 2016 IEEE
DOI 10.1109/MS.2016.19

Section IV describes the framework components. Examples are presented in Section V. We conclude the paper and outline our future work in Section VI.

## II. Motivation and Related Work

### A. Motivation

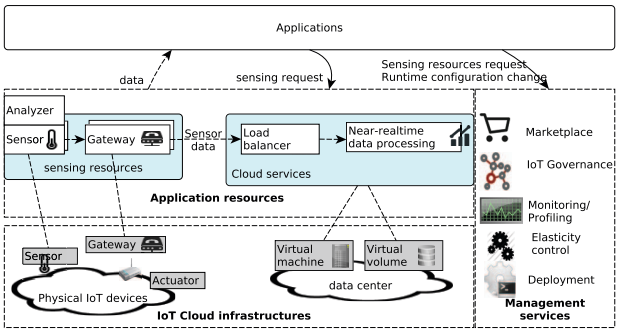


Fig. 1: Applications, sensing resources, and management services in IoT cloud systems

Let us consider a use-case of predictive maintenance of chillers in smart cities, which combines large-scale distributed IoT sensing resources and cloud-based services [3]. Shown in Fig.1, the IoT cloud infrastructures characterize basic elements in physical IoT devices and data centers, such as hardware sensors, gateways, and virtual machines. They are available for different applications and sensing resources. The *IoT sensing resources* include, e.g., software sensors, actuators, lightweight analyzer, and gateways on top of IoT infrastructures[1]. We also have *cloud services*, such as load-balancer and near-realtime data processing, running on the cloud data center, that handle heavy workload such as analyzing sensory data and processing events. Sensing resources and cloud services are *application resources* that can be instantiated on-demand. Several *Management Services* support deployment, control and monitoring of these resources in IoT cloud systems. For examples, at runtime, IoT sensing resources can be reconfigured to increase the updating frequency and to change the communication protocol, or we can deploy more sensors for more fine-grained data during emergency situations. Furthermore, cloud services can be reconfigured to handle the increasing amount of incoming data from the IoT elements.

The application in this use case is to perform predictive maintenance by deploying sensing resources, in particular sensors and lightweight analyzers, and controlling these resources to collect required information. The important aspect is that neither sensors are statically deployed and executed nor sensor capabilities are unchanged during the application lifetime, due to the need to have different sensing information at runtime; and this aspect is supported by recent advanced virtualization and software-defined techniques. Applications face great challenges in capturing and collecting the information about sensing resources due to the distributed, heterogeneous and dynamic environments. For example, if an application wants to reconfigure sensing resources, then the application needs to know, e.g., the capability to change the communication protocol between a sensor and a gateway, or the capability to scale out cloud services upon high workload of the near-realtime analysis. Sometimes sensing resources do not provide a specific capability by default, but existing management services can provide the capability, e.g. on-the-fly deployment or deactivation of sensors.

Therefore, applications and their developers need support from extensible and scalable frameworks for providing information of complex IoT sensing resources and their dependent resources. Nevertheless, developing such frameworks face several challenges: (i) identifying and correlating resources instances managed by different management services, (ii) capturing diverse types of information of about sensing resources and their relationships in a unified data model, and (iii) dealing with both horizontal and vertical complexity and dynamism of the system due to the large-scale and inherent multiple software stacks.

### B. Related work

There is a large number of research publications in wireless sensor network (WSN) deployment and information management [8]. Although sensing resources in WSN are dynamic, the execution model and infrastructures are very different from our work, where we focus on sensing resources in IoT infrastructures supported by cloud data centers. The key difference is due to the recent development of software engineering, virtualization and software-defined techniques as well as business models of utilizing sensing resources as a service. In the following, we examine our main related research:

*IoT Sensing Resource Deployment*: Several papers support static sensor deployments. Therefore, sensing resources information is mainly about resource status (for accessing resources) from pre-deployed management services which do not support information for configuration and control of sensing resources. There are several papers discussed about IoT deployment. In [9], the authors present a comprehensive framework for deploying IoT resources. In ThingStore [10] deploying things from marketplaces is also discussed. Obviously collecting information about sensing data is also related IoT deployment. However, our work differs from these papers as we focus on managing dynamic information about sensing resources, not on deploying the resources.

*Information management:* Endo et al. [11] introduce concepts and frameworks of information system that supports capturing life-cycle of cloud-based services. Johannes et al. [12] address the information collection problem and build a knowledge-base to support the development phase. Nakamura et al. [13] present method to manage IaaS resource for the adaptive configuration by using ontology. However, these studies focus on a narrow view to support single software stack for cloud services, either application level or infrastructure level, and only for development phase. In this work, we aim to provide information of multiple system stacks at runtime with a focus on the lifecycle of IoT resources. In [14] a federated model of IoT discovery using ontology is proposed. However, it does not focus on different phases in the IoT lifecycle as well as the information collectors for on-demand sensing resources.

[1] In this paper, we refer to sensors, actuators and gateways as software ones which may interface to their hardware counterparts

*Information monitoring and collection:* Andreolini et al. [15] introduce an algorithm to monitor large-scale systems, that adapt the monitoring frequency for resource and cost. Zang et al. [16] introduce architectures of information services to collect and query information from heterogeneous infrastructures. Our approach does not aim to propose a monitoring solution but to interface to monitoring information and provide required information and APIs to enable the access to on-demand sensing resources for applications to control sensing resources to optimize application functions with changing application requirements.

## III. Capturing Sensing Resources Information

### A. Information collection

As mentioned, in our work, we focus on the software layer in IoT cloud systems. All our sensing resources are software which can interface to hardware sensors, actuators, gateways, network and virtual machines. Studying from the use-case in Section II, we focus on the main types of sensing resources described in Fig.2. These sensing resources can be instantiated through the deployment of their artifacts in repositories and marketplaces (e.g., sensors, lightweight analyzer, and actuators) or through the allocation of infrastructure capabilities (e.g., network). This is possible today due to change in software development, the business model (sensors-as-a-service) and virtualization techniques (where different sensors can be deployed into virtual environments).

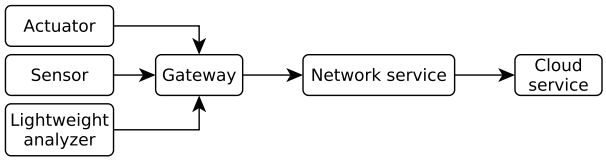


Fig. 2: Sensing resources and their dependencies

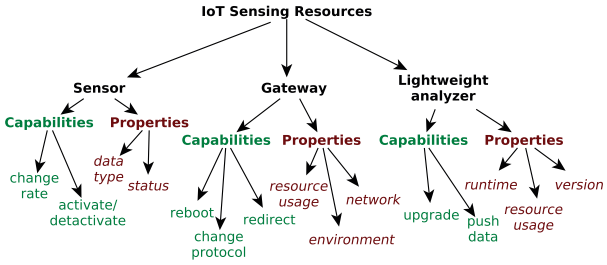


Fig. 3: Example of types of sensing resources information

To support application requirements, we need to collect different types of information that the application can use to configure and control sensing resources. Shown in Fig.3, we classify the information collected for sensing resources into: (i) *Properties* describing the states and metadata of the systems, and (ii) *capabilities* describing actions that can be performed for configuring particular sensing resources. These types of information are provided by different management services. For example, a sensor can have a capability of changing data rate in its sensing and this capability can be captured by the deployment service. In order to increase data rate in sensing, we can query the deployment service to see if an instance of the sensor can be reconfigured with different sensing rates.

Collecting the types of information of IoT sensing resources spans different phases of development and operation and the information from these phases must be correlated in order to provide sufficient information about IoT resources for any application. Fig.4 shows related properties and capabilities of a service along the development and operation. It requires various collecting methods to interact with different kinds of information sources.

Certain types of capabilities and properties can be provided by the resource providers at the development time and usually published on some repositories (e.g. Chef cookbook[2], Docker hub[3]), marketplaces (e.g. Amazon Web Services' Marketplace[4] or ThingStore [10]), or custom metadata services (e.g.WSO2 Governance Registry[5]). This is due to the recent trend in IoT marketplaces [17]. From our experience, this way is also used for provisioning sensing resources in industrial cloud-based M2M (Machine-to-Machine) and building management, as different types of customers require different sensors and lightweight analyzers for different Things and these sensors and analyzers are developed by different third parties/providers. Normally, these repositories and marketplaces provide structured data that can be understood and obtained automatically via existing adaptors [12].

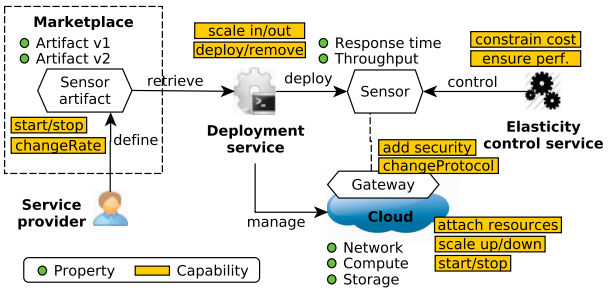


Fig. 4: Collect capabilities and properties of sensing resources – an example with sensors

At runtime, further capabilities and properties can be collected dynamically from various management services. In general, the management services can expose the information that they have, e.g. the *states* of configuring a sensor (e.g., deployed, active, and inactive) and the capabilities to change these states. Some types of information are provided via domain specific models and require tight collaborations with the management services to extract the needed information, e.g. deployment states from the deployment service are strongly dependent on concrete deployment services in specific clouds and IoT systems. The management services can also capture relationships between sensing resources, such as the deployment service knows the number of sensor instances sharing the same gateway, or the monitoring service usually records the throughput between a gateway and a cloud service.

[2]https://supermarket.chef.io/cookbooks
[3]https://registry.hub.docker.com/
[4]https://aws.amazon.com/marketplace
[5]http://wso2.com/products/governance-registry/

### B. Capturing configuration properties and capabilities

The information collected from different management services is usually in different formats, containing inconsistent property names and providing different types of capabilities. Tables I, II and III show examples of diverse capabilities. In order to provide usable information for the configuration, we devise a unified model to represent the service properties and capabilities. This allow us to remove the complexity of parsing multiple information types and allow the extensibility to integrate with new management services.

TABLE I: Capabilities of generic sensing resources

| Generic types | Capabilities |
|---|---|
| Gateway | change protocol, connect to gateway, run lightweight app. |
| Sensor | change frequency, change protocol, connect to gateway |

TABLE II: Capabilities of specific cloud services

| Specific types | Capabilities |
|---|---|
| HAProxy | change max connection |
| Event Processing | forward data to queue/database |
| Cassandra cluster | scale out/in |
| RabbitMQ | modify policies, change environment variable |

TABLE III: Capabilities of management services

| Management services | Capabilities |
|---|---|
| Deployment service | deploy, resolve dependency, manage cloud resources |
| Elasticity control service | start/stop control, replace control policies |
| IoT Governance service | add primitive capabilities, invoke capabilities in batch |
| Monitoring service | change frequency, update monitoring metrics |

To develop the unified model, we elaborate the concepts from previous work [18]. In our model, we consider any sensing resource as a "service" in an abstract way. This term "service" also aligns with existing concepts of sensor-as-a-service, analytics-as-a-service (e.g., for lightweight analyzer) and sensing-as-a-service [6], [19]. Thus, in the model we use the *Service unit* for generalizing different types of sensing resources and cloud services. Fig.5 depicts our unified model that aims to support the dynamic configuration and to fill the gap between cloud models for static description (reflected via Service Artifact) and data models for dynamic information (reflected via Service Instance). The *Capability interface* provides a same way to access different operations and can be realized by multiple *execution models*, thus enables the configuration to select alternative implementation depending on particular situations. For example, a service exposes reconfiguration functions via scripts, which can also be executed via the API of the deployment service with some additional qualities, e.g. increasing the fault tolerance, but reducing the performance. The domain specific information are captured via service's *category* and *domain properties*. Fig.6 shows an example of capabilities collected for a sensor instance. These capabilities are linked from both static and dynamic information.

A service can evolve through three phases: development, deployment and operation which different information can be captured. The *offered service*, which is not initiated, contains a set of artifacts that require for the default *deploy* and *undeploy* capability. The offered service is initiated at deployment time and become a *service instance*, which contains runtime information, e.g. state. The relationships between service instances are captured via *service topologies*.

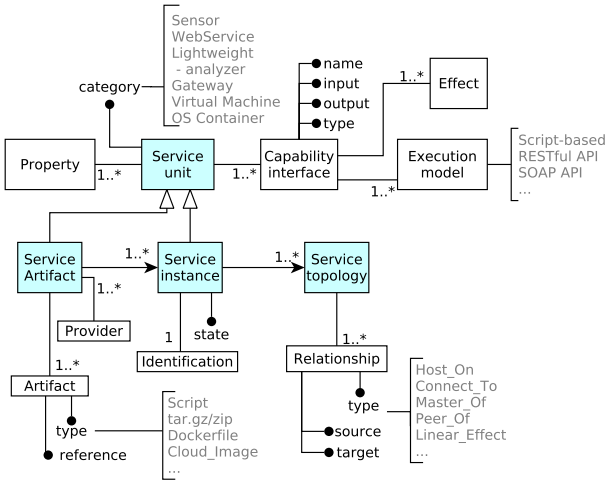


Fig. 5: Unified model of sensing resources and cloud services

### C. Identifying sensing resources instances

Within an IoT cloud system, sensing resources and cloud services are dynamically changed (e.g., deployed, migrated, and deactivated). Therefore, we must be able to identify right instances of service units in order to provide accurate information for reconfiguring service instances at runtime. Cloud providers and management services use different identity (ID) schemes to identify service instances, which called *domain ID*, e.g. IP address or Object identifier (OID) [20]. It is impossible to have a single ID scheme because service instances are managed independently. Some identification schemes include information that we can interpret from, such as the ID of a system service is the pair of IP address and port (TCP socket), or the ID of a web resource can be a hierarchical structure (e.g. URI, URL). However, many service providers use UUID for identifying services, which cannot be interpreted. For accessing the information by a unified manner, we need a single identification for each service instance, so-called *global ID*. In our approach, a global ID is mapped to a list domain IDs, enables to refer to all information about an instance via multiple sources. Therefore, we need a mechanism to detect that a set of domain IDs belong to a same instance or not, afterward we can assign to that instance a global ID.

In our approach, management services share some non-unique attributes of service instances with other management services. Based on that, we can correlate the information collected from management services into service instances, by applying existing information matching and similarity measure functions. This mechanism requires the agreements between management services about the shared attributes, but does not need the interaction between management services. We use the collection processes (Section III-A) to extract shared attributes from sensing resources instances. Fig.7 shows an example of managing identifications for multiple management services.

Our approach relies on matching functions which may bring uncertain information about sensing resources. Currently, we are investigating how to incorporate uncertainty about sensing resources [21] into our work.

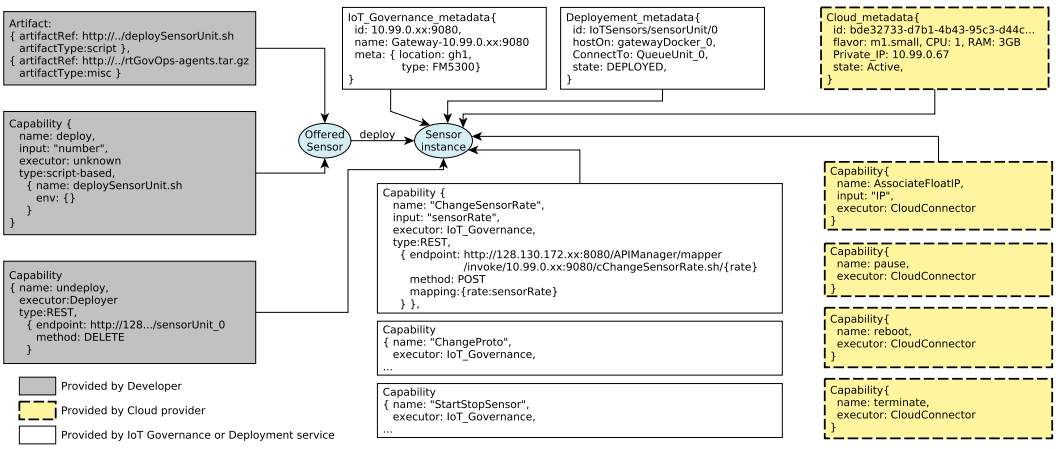


Fig. 6: An example of capabilities information

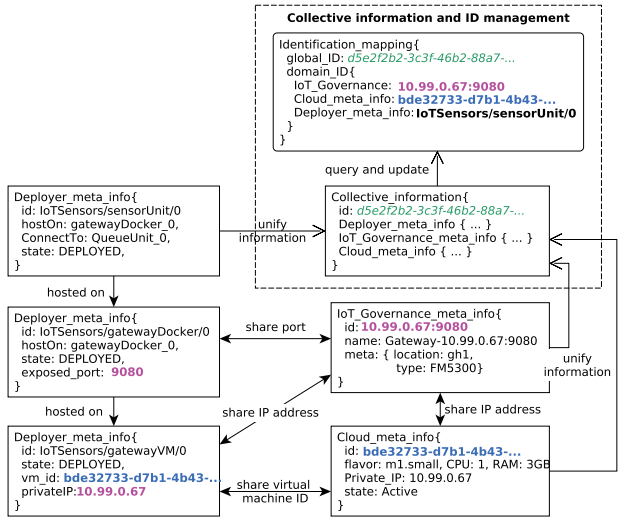


Fig. 7: Example of identification management

### D. Providing capabilities information

As the information is collected from many places, this could cause substantial overheads. Thus, an application needs only meaningful information for a particular configuration. For example, we need to query for all the sensors whose capability can change the sensing *rate*. To obtain correct information, one needs to be able to query and subscribe for the particular information it is interested in. We design the query format as $Query = (category, \{rules\}, \{capability\})$. An example of a query in JSON is as follows:

```
{
    "category": "SENSOR",
    "rules": ["location:EQUAL:bld1", "type:EQUAL:motor"],
    "hasCapabilities": ["ChangeSensorRate", "deploy"]
}
```

The result of the query is a set of service instances. Fig.6 shows fragments of information collected from different management services. Such information can be parsed and utilized within the application itself or by other tools to manage sensing resources.

## IV. Configuration Information as a Service

### A. Loose-coupling architecture

IoT cloud systems may be extended, shrunken or migrated, sensing resources and information gathering components are distributed and dynamically changed. We design our ELISE (Elastic Configuration Information as a Service) to simplify the management of information for sensing resources by dealing such dynamism. Fig.8 overviews the architecture of ELISE.

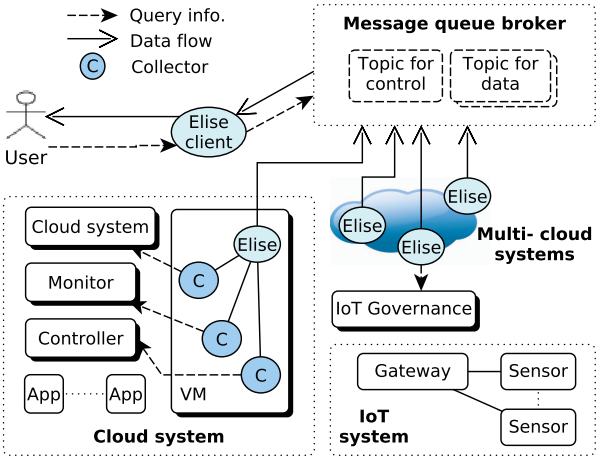


Fig. 8: Distributed information management

ELISE contains a set of *collectors* to interface with various management services to collect information. Collectors are callable modules (e.g. in the form of a class or an executable program) that connect to the source of information, query the source, and translate data to the ELISE's model. ELISE manages collectors and calls appropriate ones only when needed, allowing us to dynamically add or remove collectors at runtime.

For deploying in a large scale setting, ELISE fully leverages the message queue and publish/subscribe communication. This enables ELISE to be agnostic to others and independent with the network topology, also provides the flexibility in

managing information, e.g. we can add a new ELISE easily when the system extends or a new cloud provider is used. This also eliminates the need of managing multiple ELISEs, single-point-of-failure and bottleneck. ELISE can be deployed closer to the information sources for faster and more reliable data collection, e.g. an ELISE instance can run with a management service on the same VM.

### B. Protocols for transferring and querying information

Because ELISE needs to integrate to different collectors, we need a protocol to send different kinds of commands to help ELISE to optimize its work, e.g., to manage multiple ELISE and many message transmissions at the same time. We use the following message structure for commands:

$$MSG = (CMD, from, topic, feedback, payload)$$

where:

- CMD: the message type (discover, query, response).
- from: the ID of the ELISE service sending message.
- topic: the topic to broadcast a message.
- feedback: the topic that the response is sent.
- payload: the content of a query or a response.

At the beginning, an ELISE instance is configured to subscribe to a *control topic* to listen to the messages from the others. The control topic is used for distinguishing different groups of ELISEs. The set of (from, topic, feedback) enables to distinguish between concurrent messages, also to drop the duplicated messages created by message queue platform. For the management ELISEs, we can broadcast a *discover* message and get back the number and description of all ELISEs. For querying information, all the ELISEs will receive the *query* and update information through the feedback topic.

### C. Deploying and connecting data collectors

Because of the on-demand deployment of sensing resources, *collectors* are also deployed on-demand on IoT cloud infrastructures, and are connected each other to establish the system providing sensing resources information. This poses several challenges. First, we must deploy the collectors on suitable targets, e.g. to push artifacts to the right gateways. For this, we undergo two steps: (1) collect and understand the information of the infrastructure via a management service or clouds, (2) by means of existing deployment techniques [22], we can deploy the components depending the target environment. After the deployment, the collector publishes a message to register itself to ELISE and open a communication channel via message queue. Second, the source of information must be configured for each collector, e.g. the endpoint of web service where information is exposed. The collector can be implemented by ad-hoc way to interact with predefined resources. In other way, the manual configuration is pushed to the collectors from the client via ID of the collectors.

### D. Prototype

ELISE[6] exposes a set of RESTful APIs to get the requests from applications/users and to retrieve data from collectors. We use Neo4j[7] graph database for storing and querying services based on their relationships. For the communication among ELISE components, we have implemented adaptors for MQTT[8] and Apache Kafka[9], which can be configurable when running ELISE. Kafka is used to provide higher throughput for heavy messages transferred within ELISE. We implemented a set of collectors for clouds, including OpenStack by jClouds[10] and Flexiant by JADE[11]. We have developed collectors to interface to different management services in the iCOMOT framework[12], including a Deployment Service, an Elasticity Controller, a Monitoring Service and an IoT Governance Service. These management services provide enough information reflecting various actions of sensing resources at runtime.

## V. Examples

This section will show how ELISE framework can enable the unified management, the extensibility for different system scales and the flexibility for the sensing resource dynamicity.

### A. Providing configuration information

In this example, we demonstrate our work by emulating an IoT cloud system for sensing chiller operations in buildings in smart cities. The system has two parts: sensing resources deployed in IoT devices and cloud services in data centers. The cloud services are non-scalable components (ActiveMQ, HAProxy and Cassandra seed) and horizontal scalable components (Local Processing, Event Processing and Cassandra Node). The sensing resources include sensors and gateways with different communication protocols to the cloud. The IoT cloud system for sensing chillers contains complex cloud services, which are managed by several management services on both cloud infrastructures.

All of software for sensing resources and cloud services are implemented with realistic functions but we experimented their operations in an emulated environment – not a production environment (shown in Fig.9). We deployed cloud services in Flexiant FCO – a public cloud. Our sensors utilized real data set from an industrial partner and replayed the data. To emulate sensing resources, we use another private cloud (`DSG OpenStack`) with lightweight virtual machines (denoted by `m1.small` – 1CPU/3GB) and dockers as the environment for running sensors and gateways. Sensors are executed using *docker* on Openstack VMs. Several management services (Mgt.Serv.) are deployed on the two clouds to manage the IoT cloud system. We deploy one ELISE instance for each cloud. The two ELISE instances are configured with a set of collectors to interact with management services and cloud APIs to retrieve the information. When deploying ELISE instances, we must manually configure collectors with the endpoints of the management services or the cloud APIs.

Given a query for sensors monitoring chillers, Listing 1 shows an excerpt of the sensing resources information, which indicates a list of sensors in JSON. Information of each sensor

[6]Prototype and supplement materials: http://tuwiendsg.github.io/SALSA/elise.html

[7]http://neo4j.com
[8]http://mqtt.org/
[9]http://kafka.apache.org/
[10]https://jclouds.apache.org
[11]http://docs.flexiant.com/display/DOCS/Introduction+to+Jade+APIs
[12]http://tuwiendsg.github.io/iCOMOT

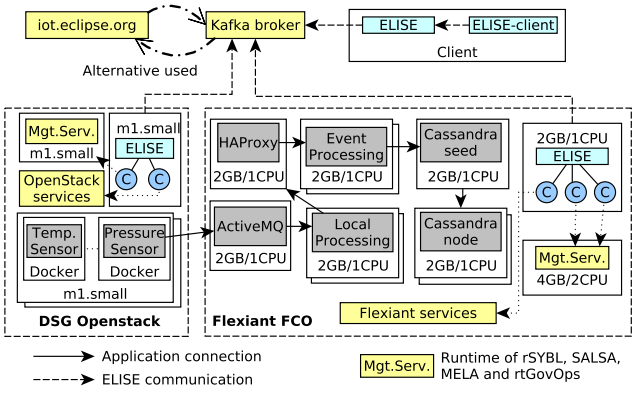


Fig. 9: Deployment of resources for sensing chiller operations

contains different IDs from management services, which are unified with a global ID (the first ID). The global ID, which is an UUID, enables the application to refer to a specific sensor via a single reference (e.g. the `id` in Listing 1). Also, the domain IDs are available for further interactions with the sensors via management services. A full view of the sensor is captured via *properties*, e.g., the state, IP address, location, and sensor types. Similarly, the *capabilities* collect all possible operations on the sensor, including the reference of how to execute the operations, such as the endpoint of RESTful services.

Listing 1: Example of a query result for sensor instances

```
[{"id": "19866cf7-6d59-4a7c-ac6d-a01f33cd9e6e",
  "name": "Chiller_motor_status",
  "category": "SENSOR",
  "properties": [
    { "name": "salsa-metadata",
      "metrics": [
        { "name":"id", "value":"Chiller/building1/motor/0"},
        { "name":"state", "value":"deployed" },]},
    { "name": "govops-metadata",
      "metrics": [
        { "name":"location", "value":"building1" },]},
    { "name": "vm-metadata", ...}}],
  "capabilities": [
    { "name": "deploy",
      "executionMethod": "REST",
      "executionModel":"{
        \"endpoint\":\"http://128.130.172.215:8380/salsa-
            engine/services/Chiller/topologies/building1/
            nodes/motor/instance-count/{quantity}\",
        \"method\":\"POST\",\"data\":\"\"}"  } ],
 //.... omitted
    { "name": "undeploy", ...},
    { "name": "StartStopSensor", ...}
    { "name": "ChangeProto", ...} } ]
[more sensors are shown here]
```

The information of capabilities is up-to-date and reflects runtime status by including the concrete endpoints of the capabilities, which is ready-to-use in a configuration. The set of capabilities is merged from different management services and cloud providers in order to provide a unified view of all the configuration operations. Using provided information, the the application or other tools can apply different strategies for particular configuration goals.

During runtime of the system, the collected information can become insufficient due to the expansions and changes of the IoT cloud system, e.g. components are migrated, new components join, or new management services involve. The information collection must be extensible to adapt with these changes, e.g., to collect new types of information that become available at runtime. To show the extensibility of ELISE, we keep the IoT cloud system running and deploy more ELISE instances to gather information. The collectors are configured and added one by one to ELISE instances when the system is running. As the result, the response of the queries (in Listing 1) becomes richer after a collector is added: the number of properties and capabilities increases when we added more collectors. The loose-coupling architecture enables ELISE to be extensible on-the-fly.

Given an IoT cloud system, this feature of providing up-to-date sensing resources information will allow applications to implement their decision on how to optimize their functions. In parallel, external utilities can be developed to control sensing resources, e.g., activating more sensors or changing communication protocols.

### B. *Enabling programming sensing resources*

An application aims to reconfigure the *rate* of sensors in order to obtain fined-grain information or to reconfigure the communication protocol between a gateway and the cloud. Because the systems are changed over time, the application does not know about management services and sensors, e.g. the endpoints of the management services, the places where the sensors are deployed, the states of the sensors. ELISE simplifies this problem by providing APIs for accessing runtime data and APIs for invoking capabilities. As described in the previous subsection, APIs for querying sensing resources can return information shown in Listing 2.

Listing 2: Example of sensor and gateway capabilities

```
{ "name": "ChangeSensorRate",
      "executionMethod": "REST",
      "executionModel":"{
        \"endpoint\":\"    http://128.130.172.199:8080/
            APIManager/mapper/invoke/10.99.0.102:9080/
            cStartStopSensor/startcChangeSensorRate/update?
            args={rate}",
        \"method\":\"GET\",\"data\":\"\"}"  } ],
//... omitted
{
      "name": "changeProtocolMQTT",
      "executedBy": "GovOps",
      "executionMethod": "REST",
      "executionModel": "{\"endpoint\":\"http:
          //128.130.172.199:8080/APIManager/mapper/invoke
          /10.99.0.102:9080/cStartStopSensor/cChangeProto/
          upd
ate?args={protocol}",\"method\":\"GET\",\"data\":\"\"}"  } ]
    ,
      "parameters": [],
      "effects": []
    }
```

The capabilities to change the sensors' rate or communication protocols can be invoked based on the information about execution methods and models in the description. In this case, corresponding management services are responsible for invoking such capabilities. For example, Listing 3 presents an excerpt of code for changing the rate of sensors.

Listing 3: Example of programming a logic to change protocols

```
String endpoint = ".../rest/elise/";

// the 1st proxy to manage the queries
manageProxy = create(endpoint, Communication.class)
```

```
query = new EliseQuery(ServiceCategory.Gateway)
  .hasRule("location", "building1", OPERATION.EQUAL);
queryID = manageProxy.querySetOfInstances(query);
manageProxy.getQueryStatus(queryID);
//....
// the 2nd proxy to query and invoke capability
localProxy = create(endpoint,UnitInstance.class);
instances = localProxy.queryUnitInstance(query);
for (UnitInstance instance : instances) {
  c = instance.getCapabilityByName("changeRate");
  if (c != null) {
    CapabilityMng.execute(c, new String[]{"5"});
  } }
```

## VI. Conclusions and Future Work

Given advanced techniques allowing us to deploy and control sensing resources on demand in IoT cloud systems, it is crucial that, in order to simplify the access and control of sensing resources, we must be able to deal with dynamic and complex types of information about sensing resources through different deployment and operation phases. In this paper, we have identified main types of capabilities and properties that need to be captured for dynamic sensing resources. We approach the above-mentioned challenges by designing a scalable and extensible framework that utilizes various types of data collectors to interface to APIs of different management services and sensing resources repositories/marketplaces. This allows us to avoid tightly integration with various specific systems by going inside these systems and changing them, which is not scalable, if not impossible, in the IoT world. Instead, we have addressed scalable protocols to gathering information, identification integration, information gathering through different phases of IoT and cloud services. Our prototype – ELISE – showed that extensibility and flexibility are key issues in order to enable diverse types of configuration information.

We are currently working on more types of information and improve quality of information. Furthermore, we focus on extending and experimenting our framework for other application domains, such as sensing in Geo Sport applications[13], and uncertainty information about sensing resources.

## References


[1] A. Botta, W. de Donato, V. Persico, and A. Pescape, "On the Integration of Cloud Computing and Internet of Things," in *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*, pp. 23–30.

[2] E. D. Simmon, K. sook Kim, E. Subrahmanian, R. Lee, F. J. de Vaulx, Y. Murakami, K. Zettsu, and R. D. Sriram;, "A Vision of Cyber-Physical Cloud Computing for Smart Networked Systems," 2013.

[3] H.-L. Truong and S. Dustdar, "Principles for Engineering IoT Cloud Systems," *Cloud Computing, IEEE*, vol. 2, no. 2, pp. 68–76, Mar 2015.

[4] D. H. Phan, J. Suzuki, S. Omura, and K. Oba, "Toward sensor-cloud integration as a service: Optimizing three-tier communication in cloud-integrated sensor networks," in *Proceedings of the 8th International Conference on Body Area Networks*, ser. BodyNets '13. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013, pp. 355–362.

[5] A. Krylovskiy, "Internet of things gateways meet linux containers: Performance evaluation and discussion," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, Dec 2015, pp. 222–227.

[6] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Sensing as a service model for smart cities supported by internet of things," *Trans. Emerg. Telecommun. Technol.*, vol. 25, no. 1, pp. 81–93, Jan. 2014. [Online]. Available: http://dx.doi.org/10.1002/ett.2704

[7] S. Distefano, G. Merlino, and A. Puliafito, "A utility paradigm for iot," *Pervasive Mob. Comput.*, vol. 20, no. C, pp. 127–144, Jul. 2015.

[8] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Comput. Netw.*, vol. 52, no. 12, pp. 2292–2330, Aug. 2008.

[9] M. Voegler, J. M. Schleicher, C. Inzinger, and S. Dustdar, "A scalable framework for provisioning large-scale iot deployments," *ACM Transactions on Internet Technology*, 2016.

[10] K. Akpinar, K. A. Hua, and K. Li, "Thingstore: a platform for internet-of-things application development and deployment," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, DEBS '15, Oslo, Norway, June 29 - July 3, 2015*, F. Eliassen and R. Vitenberg, Eds. ACM, 2015, pp. 162–173.

[11] R. Schmidt, "Conceptualisation and lifecycle of cloud based information systems," in *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2012 IEEE 16th International*, pp. 104–113.

[12] J. Wettinger, V. Andrikopoulos, and F. Leymann, "Automated capturing and systematic usage of devops knowledge for cloud applications," in *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, March 2015, pp. 60–65.

[13] L. Nakamura, J. Estrella, R. Santana, M. Santana, and S. Reiff-Marganiec, "A semantic approach for efficient and customized management of IaaS resources," in *2014 10th International Conference on Network and Service Management (CNSM)*, Nov 2014, pp. 360–363.

[14] P. Gomes, E. Cavalcante, T. Rodrigues, T. Batista, F. C. Delicato, and P. F. Pires, "A federated discovery service for the internet of things," in *Proceedings of the 2Nd Workshop on Middleware for Context-Aware Applications in the IoT*, ser. M4IoT 2015. New York, NY, USA: ACM, 2015, pp. 25–30.

[15] M. Andreolini, M. Colajanni, M. Pietri, and S. Tosi, "Real-time adaptive algorithm for resource monitoring," in *9th International Conference on Network and Service Management (CNSM)*, Oct 2013, pp. 67–74.

[16] T. Zang, W. Jie, T. Hung, Z. Lei, S. Turner, and W. Cai, "The design and implementation of an ogsa-based grid information service," in *Web Services, 2004. Proceedings. IEEE International Conference on*, July 2004, pp. 566–573.

[17] C. Perera, C. Liu, and S. Jayawardena, "The emerging internet of things marketplace from an industrial perspective: A survey," *Emerging Topics in Computing, IEEE Transactions on*, vol. 3, no. 4, pp. 585–598, Dec 2015.

[18] G. Copil, D. Moldovan, H.-L. Truong, and S. Dustdar, "Multi-level elasticity control of cloud services," in *Service-Oriented Computing*, ser. Lecture Notes in Computer Science, S. Basu, C. Pautasso, L. Zhang, and X. Fu, Eds. Springer Berlin Heidelberg, 2013, vol. 8274, pp. 429–436.

[19] J. Zhang, B. Iannucci, M. Hennessy, K. Gopal, S. Xiao, S. Kumar, D. Pfeffer, B. Aljedia, Y. Ren, M. Griss, S. Rosenberg, J. Cao, and A. Rowe, "Sensor data as a service – a federated platform for mobile data-centric service development and sharing," in *Proceedings of the 2013 IEEE International Conference on Services Computing*, ser. SCC '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 446–453. [Online]. Available: http://dx.doi.org/10.1109/SCC.2013.34

[20] G. Roussos and P. Chartier, "Scalable id/locator resolution for the iot," in *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, Oct 2011, pp. 58–66.

[21] M. Zhang, S. Ali, T. Yue, D. Pradhan, B. Selic, O. Okariz, and R. Norgren, "An uncertainty taxonomy to support model-based uncertainty testing of cyber-physical systems." Simula Research Laboratory,, Tech. Rep. TR 2015-3, 2015.

[22] D.-H. Le, H.-L. Truong, G. Copil, S. Nastic, and S. Dustdar, "SALSA: A Framework for Dynamic Configuration of Cloud Services," in *6th International Conference on Cloud Computing Technology and Science*, Dec 2014.


[13] http://www.u-test.eu/use-cases/