On Engineering Analytics for Elastic IoT Cloud Platforms

Hong-Linh Truong(✉), Georgiana Copil, Schahram Dustdar, Duc-Hung Le, Daniel Moldovan, and Stefan Nastic

Distributed Systems Group, TU Wien, Vienna, Austria
{truong,e.copil,dustdar,d.le,d.moldovan,s.nastic}@dsg.tuwien.ac.at

Abstract. Developing IoT cloud platforms is very challenging, as IoT cloud platforms consist of a mix of cloud services and IoT elements, e.g., for sensor management, near-realtime events handling, and data analytics. Developers need several tools for deployment, control, governance and analytics actions to test and evaluate designs of software components and optimize the operation of different design configurations. In this paper, we describe requirements and our techniques on supporting the development and testing of IoT cloud platforms. We present our choices of tools and engineering actions that help the developer to design, test and evaluate IoT cloud platforms in multi-cloud environments.

1 Introduction

Recent complex business and societal requirements, e.g., in smart cities, cyber-physical systems, building information management, and logistics [7,16], have fostered the integration between the Internet of Things (IoT) and cloud services. Generally speaking, an IoT cloud platform includes several "Things" connected to cloud services in data centers using various software and layered protocols, as intensively discussed in [5–7,15]. Technically, such platforms are realized by having sensors and actuators interfacing 'Things"; these sensors and actuators are used to monitor and control "Things" (sensors/actuators can also be "Things"). These sensors and actuators are connected to and/or accessible from cloud services via gateways or intermediate nodes. Inside IoT gateways, different software components are used to relay sensoring data to data centers, to (pre)process sensoring data, or to execute commands from data centers to Things. There are various forms of integration of the IoT part (sensors and gateways at the edge) and the cloud part (cloud services in data centers), such as shown in [5,7,16], creating so-called IoT cloud platforms. Such IoT cloud platforms include mixed functionalities from typical cloud services and IoT elements that require novel engineering techniques [20]. Clearly, development and testing of such IoT cloud platforms are very challenging due to several reasons. First, it is not easy to

This work was partially supported by the European Commission in terms of the CELAR FP7 project (FP7-ICT-2011-8 #317790) and the U-Test H2020 project (H2020-ICT-2014-1 #645463).

Q.Z. Sheng et al. (Eds.): ICSOC 2016, LNCS 9936, pp. 267–281, 2016.
DOI: 10.1007/978-3-319-46295-0_17

specify and control complex topologies of sensors and gateways in the IoT side that emulate a real-world deployment, e.g., for equipment management in a city. Second, when storing and/or processing IoT data in the cloud, cloud services have to be designed in such a way that they support elasticity/change in the IoT side, enabling efficient operation of the whole IoT cloud platform in an end-to-end manner. However, it is challenging to monitor and control software components and services spanning across multiple clouds, including those at the edge of the clouds, in order to adapt the varying load, cost and performance. Therefore, it would take a long time to design IoT cloud platforms, as it is still hard to test configurations and features as early as possible, especially when we need to combine all running pay-per-use services in the cloud with IoT elements.

As integrating cloud software services and IoT in complex, large-scale scenarios is very challenging, various works have been focused on cloud engineering techniques for IoT cloud platforms [2,7,9]. Nevertheless, we still face great engineering difficulties due to the lack of suitable tools to test and evaluate complex designs of IoT cloud platforms. In our previous work, we have presented a demo of the iCOMOT framework [19] as a toolset for simplifying the management of IoT cloud systems. In this paper, we detail requirement analysis, designs, and engineering actions for iCOMOT which can help accelerating the development of IoT cloud platforms. Relying on fundamental concepts of virtualization and elasticity for both IoT and cloud resources, we utilize different tools to speed up the development of IoT cloud platforms. With our solutions we help reduce complexity and effort of IoT cloud platforms development by leveraging (i) suitable elasticity engineering techniques for cloud services, which already offer several common features for executions, data, computation and networks, (ii) open data sets, which can be used to emulate sensor behaviors, and (iii) appropriate integrated engineering tools for performing deployment, control and analytics tasks for IoT cloud platforms. In this paper, we also illustrate the usefulness of our techniques through a case study for predictive maintenance.

The rest of this paper is organized as follows. Section 2 presents our motivation. Section 3 describes our iCOMOT toolset. We present requirements and our design and engineering actions in Sect. 4. We present a case study in Sect. 5. Related work is discussed in Sect. 6. We conclude the paper and outline our future work in Sect. 7.

2 Motivation – IoT and Cloud Integration Support

2.1 IoT and Cloud Integration Models

Developers of IoT cloud platforms can have varying goals:

– *Goal 1:* developers might need only to develop IoT elements (e.g., sensors, actuators and gateways) for a specific customer and to connect these IoT elements to existing cloud services. In this case, they might just want to develop and test a set of sensors that can be deployed into certain gateways sending the data to public cloud services.

- *Goal 2:* developers focus on only cloud services at data centers that serve IoT elements. Typically they focus on a set of complex cloud services, e.g., for data storage, complex event processing, and data analytics.
- *Goal 3:* developers want to design and test a complete IoT cloud platform for a specific customer. They, therefore, focus on both IoT elements and cloud services and on how to coordinate them in a unified view for the customer.

Numerous works support the development of either the IoT part or the cloud services for IoT [2,12,21]. However, there is a lack of tools and discussions for the development and operations of the last goal– *Goal 3*. We focus on supporting developers to concentrate on *Goal 3*, which is complex but of paramount importance for several customers, e.g., predictive maintenance of equipment, on-demand crowd sensing for safe cities [1], and sports events [3]. For such platforms, we must deal with different engineering principles outlined in [20].

2.2 Development in Distributed IoT and Cloud Systems

As the IoT cloud platform is complex, it is expected that during the development, components of the platform must be deployed in multiple IoT and cloud systems. For this, we must have a set of connectors allowing the developer access to clouds and IoT specific systems so that the developer can deploy testing infrastructures and run tests across clouds and IoT specific systems. We also need to deal with different mechanisms of controlling virtual resources and different performance settings (e.g., expected time for allocating a resource, expected performance for each resource). As discussed in the related work (Sect. 6), most tools either enable IoT deployment or cloud deployment; even many industrial systems support IoT and cloud services, and enable their integration, these systems support *Goal 1* and *Goal 2* separately.

Moreover, at runtime, both IoT elements and cloud services need to be controlled, monitored, and analyzed in a coordinated manner. While throughout the development lifecycle the developer would need mechanisms to emulate sensors and gateways in the heterogeneity of different clouds, in a production environment we have to do an end-to-end control of cloud services and gateways deployed across IoT networks and clouds. This has to be done in a uniform manner, as most of the times control on one end of the IoT cloud platform would affect the control on the other end (e.g., deploying new data processing services on the gateway would change the characteristics of the load on cloud software services). As discussed in the related work, this feature is missing in most toolsets.

3 Overview of iCOMOT

To support *Goal 3* in multi-IoT and -cloud environments, we design, develop and experience different tools and engineering actions to address two different main issues: (i) to provide main software components which are software-defined, deployable and configurable, and (ii) to support easy configuration, deployment, control, and monitoring in a unified manner. To achieve the first point, we base on two main emerging research directions:

– software-defined IoT units: sensor and gateway components are considered as units that can be composed and controlled via software-defined APIs.
– cloud-based elastic services: they are common cloud services offered by different providers. To enable the elasticity, some services will be associated with elasticity capabilities.

To achieve the second point, we build a toolset to enable elasticity control developments. Figure 1 describes how we leverage our existing tools and our engineering actions to develop IoT cloud platforms. Cloud providers and third-party developers can design and provide several components and services that will available through PaaS/IaaS and public repositories/marketplaces. A developer will utilize these services and components and to develop her/his own services and components. Then s/he can utilize various tools to support service deployment, configuration, analytics and control to test and evaluate her/his designs.

To this end, we have demonstrated the iCOMOT framework to support the developer to develop and test different configurations and runtime behaviors of IoT cloud systems. iCOMOT (http://tuwiendsg.github.io/iCOMOT/) includes several individual research tools for configuring, deploying, monitoring and controlling IoT cloud platforms, such as SYBL, SALSA, MELA and GovOps [19]. We connect several common repositories and together with these tools to support automation of IoT cloud platform configurations deployment. Furthermore, several sensors with data sets are also provided. The following sections, we will explain main insightful engineering actions, designs and experiences.

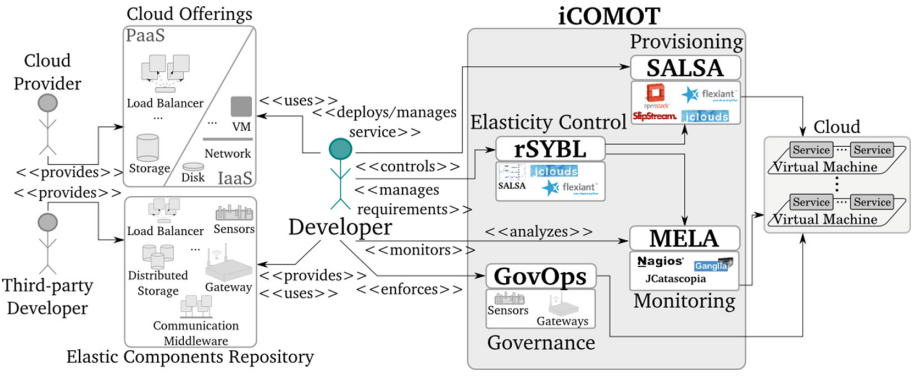

Fig. 1. Engineering tools and actions

4 Design and Engineering Actions

4.1 Software Components for IoT Cloud Platforms

IoT Units and Cloud Services Repository. *Requirements:* Existing cloud APIs already allow us to invoke suitable cloud services at data centers and

to emulate execution environments (e.g., lightweight virtual machines for emulated IoT gateways). However, they do not support the IoT side well, such as, modeling, configuring, deploying and testing different types of sensors, gateway software components, and libraries for cloud connectivity.

Solutions: To model and capture IoT unit capabilities and configuration with software artifact, we leverage the concept of IoT cloud units [14] to successfully model suitable IoT resources and enable some useful behaviors, such as dynamically changing communication protocols between IoT gateways and cloud services at runtime. As IoT units can be provided by different developers and providers, we support access to different component repositories for existing IoT units, such as gateways execution environment, virtual network routers and communication protocols for cloud connectivity. These repositories can be leveraged by well-developed technologies, such as Docker Hub, Git-based repositories and Maven, and IoT marketplaces, such as [2,4,21].

Software Sensors. *Requirements:* One important type of IoT software units are sensors. For development and testing, it is crucial to have emulated sensors whose behavior is similar to real sensors but with advanced features to allow us to easily control the sensors. An emulated sensor just takes time series datasets, e.g., obtained from industrial real systems, and replays/simulates events by sending them to gateways. By leveraging different real datasets, we can instantiate different sensors by configuring these instances with different sample data and behavior models. With this way, we can emulate GPS, temperature sensors, and chiller's operation sensors, etc., for different types of IoT cloud platforms.

Solutions: In our framework, the developer can develop her/his real sensors or emulated sensors and then deposit the sensors into the repository from which they can be deployed into IoT cloud platforms. To support rapid development of the IoT side, one important issue is to have emulated sensors as executable code that can be deployed at a very large-scale in multi-cloud environments to emulate real situations, e.g., monitoring chillers in a city, in IoT cloud platforms. We also enable the users to modify configurations while the sensors are running, for simulating different workloads or for testing their application under various out-of-the ordinary circumstances (e.g., fire at a location). The developer can also design topologies of different sensors for better management and reuse. In production scenarios of an IoT cloud platform, sensors will be physically distributed at different places, while, in simulations and tests, sensors are deployed in different VMs, OS containers, lightweight machines like Raspberry Pi, or cloud data centers. To enhance interoperability and reusability of possible sensor architectures, we present the topologies of sensors by well-known description languages, such as TOSCA (https://www.oasis-open.org/committees/tosca/), for the deployment and control process.

Software-Defined Gateways. *Requirements:* In certain IoT cloud platforms, sensors can directly connect to cloud services. However, in our experience, very

often gateways are needed as intermediate nodes for handling different types of sensoring data and connectivities. We consider and support gateways as another type of IoT software/hardware units. Gateways are much more complex than sensors. For example, gateways can store information and execute some lightweight components to process sensoring data in the cloudlets model [18].

Solutions: From the architecture design perspective, we develop and provision gateways functionality by using our concept of software-defined IoT units [14]. Generally, a software-defined gateway consists of a set of dependent IoT units deployed in a virtualized environment, e.g. CentOS or Docker. These IoT units are responsible for managing data streams, controls of actuators, cloud connectivity and lightweight data storage and processing. The key point of a software-defined gateway is that it enables dynamic deployment and configuration of IoT units to handle data, control and connectivity, allowing the developer to implement IoT-side distributed data processing, such as pre-processing data in gateways and splitting streams, i.e., sending events to multiple cloud services.

Cloud Services for IoT. *Requirements:* At data centers, both cloud-offered services and custom-built software components can be used for building the IoT cloud platform. Main cloud providers, such as Amazon EC2 (http://aws.amazon.com), Rackspace (http://www.rackspace.com), Windows Azure (http://azure.microsoft.com/en-us/) or Flexiant (http://www.flexiant.com/), offer diverse types of cloud services, from infrastructure (e.g., VMs, networking, virtual storage), to platform (e.g., load balancer, message queue), to management (e.g., monitoring, backup, auto scaling), and data analysis applications (e.g., stream data processing services). However, it is challenging to combine and use such services in a coordinated mode with IoT elements, for example, to enable the elasticity coordination between IoT sensors and cloud services.

Solutions: Focusing on elastic software components, we enable developers to employ a series of off-the-shelf software components in building their elastic platforms. The most commonly used software components are load balancers (e.g., HTTP load balancer – HAProxy http://www.haproxy.org/), which enable elasticity of web servers serving platform clients. Next, distributed data storage frameworks (e.g., Cassandra – http://cassandra.apache.org/ – and MongoDB – http://www.mongodb.org/) are crucial in building elastic data ends, but must be configured and managed accordingly, as scaling a data end is usually a complex operation, implying data moving and copying. Another class of software components used for enabling elasticity is message oriented middleware, (e.g., Apache ActiveMQ – http://activemq.apache.org/), which, by decoupling the communication between components, enable seamless addition and removal of component's instances. In general, while we are not developing such common software, we focus on how to glue them using elasticity techniques for elastic IoT cloud platforms.

4.2 Deployment, Control, and Monitoring Actions

Deployment. *Requirements:* The developer has to deploy components of IoT cloud platforms very often in order to study and test them. Generally, a deployment service will have to deal with both IoT and cloud service sides. We need to deploy different types of services and to manage from single components to the entire platform configuration at runtime. Therefore, the developer has to prepare at set of deployment artifacts in the repository including the dataset, configuration script, software artifacts. The APIs and information for accessing IoT and cloud infrastructures must also be prepared. We witnessed that these complex tasks cannot be done by a single tool, but multiple tools and connectors to different clouds, orchestrated by a centralized service.

Solutions: Currently we can describe deployment descriptions using various format such as TOSCA, HEAT (https://wiki.openstack.org/wiki/Heat) or AWS CloudFormation (http://aws.amazon.com/cloudformation/), Juju (https://juju.ubuntu.com), Cloud Foundry build packs (http://docs.cloudfoundry.org/buildpacks/custom.html). We use TOSCA intensively as it provides a generic model to define cloud services and support loose-coupling relationships between multiple component types, which makes the description independent from cloud providers and deployment tools. However, we experienced that TOSCA description is complex to create, maintain and process, so it requires comprehensive and easy-to-use tools for users and developers, especially for deploying IoT units into gateways. For the application provider who just wants to deploy the services or sensors, a description tool with GUI that hides the low-level information is more convenient. With an end-to-end aspect, our deployment tool copes with different levels of deployment, including requesting cloud provider resources, configuring virtual machines, middleware and dependencies, and deploying artifacts.

Elasticity Analytics. *Requirements:* For analyzing the elasticity change of the IoT cloud platform (e.g., scaling in/out cloud services and sensor instances), elasticity analytics will be deployed at different parts of the platform to provide different performance and elasticity metrics. An elastic IoT cloud platform would have elasticity requirements defined over its components, based on which intelligent controllers can analyze its behavior and take appropriate actions. Due to the potential complexity of IoT cloud platforms, the developer might not know such requirements for all platform components, especially reflecting the dependencies between the IoT part and the cloud part. For example, a developer might not understand the cloud storage performance required to fulfill the requirement of activating more sensors.

Solutions: To custom IoT cloud platform-specific analytics, we follow two different approaches: (i) bottom-up: common built-in metrics are structured and analyzed automatically, providing an overview over the platform's elasticity, and (ii) top-down: the platform developer can define custom, potentially domain-specific, metrics and analytics functions. Thus, we provide a complete end-to-end view

over behavioral limits of the platform to enable the developer to refine the platform, and improve its control strategies. Especially, it is crucial to define an analytics function, which, based on supplied requirements, records encountered bounds on the monitored metrics not targeted by user requirements, in which the developer requirements were respected.

Elasticity Control. *Requirements:* We need to enable elasticity for various parts of the IoT cloud platforms, such as sensors, gateways and cloud services, during the development and operation. This means that elasticity control mechanisms and tools must work across sub-platforms for design, test and operation purposes and must interface various protocols (e.g., REST, MQTT, ssh + bash execution) used to change software components. Moreover, most developers would be interested in specifying abstract, high-level requirements (e.g., not focused on system-level metrics, controlling the software service as a whole).

Solutions: For sensors, developers could control the behavior of sensors (e.g., data reading frequency), to which gateway a sensor connects as well as the protocols between by gateways and sensors. At gateways, developers could control the number of sensor connections, the amount of data which is stored locally considering various constraints (e.g., the gateway has very limited computational power, memory and space). Moreover, we can add/remove various components for locally processing information, or change their sensitivity. For cloud services in an IoT cloud platform, we can support various known control mechanisms: (i) virtual infrastructure capabilities (e.g., add/remove virtual machines, network interfaces, disks), (ii) platform specific capabilities (e.g., start/stop web server, deploy/undeploy service in existing web server), or (iii) application-specific capabilities (e.g., using API offered by cloud services developers). Each of these can be enforced separately or grouped into complex control processes. However, elasticity setup cannot be completely automated, and completely application-independent. When developers need more advanced elasticity controls, they can encapsulate them into their application-specific control mechanisms (e.g., use a web server deployed together with a new configuration to result into other performance/cost characteristics). For controlling elasticity, we enable interaction-based control to empower the developers with refining their control strategies, considering the evolution of the service at runtime.

5 Experiments

5.1 Case Studies

Let us consider a scenario in which a predictive maintenance company would like to focus on predictive analytics for chillers in a city. The company wants to reuse/rent as much as possible IoT cloud infrastructures so that the company will focus on deploying its sensors, gateways, and cloud services. These sensors, gateways and cloud services establish the company's IoT cloud platform. The IoT cloud platform includes gateways at the IoT part and cloud services at the data center. All of them are virtualized services, meaning that they can be

deployed, configured and used on-the-fly. The predictive maintenance company will need features from the IoT cloud platform provider, which provides the right configuration of the IoT cloud platform for the predictive maintenance company. The IoT cloud platform can offer features for a predictive maintenance company which monitors chillers and performs data analytics and maintenance tasks. In this case study, we will focus on the case that the predictive maintenance company wants to buy services from an IoT cloud platform provider to create a configuration of its own elastic IoT cloud platform. Then the company develops and tests different sensors which connect to its elastic cloud services to have a complete IoT cloud system for gathering data to support data analytics[1].

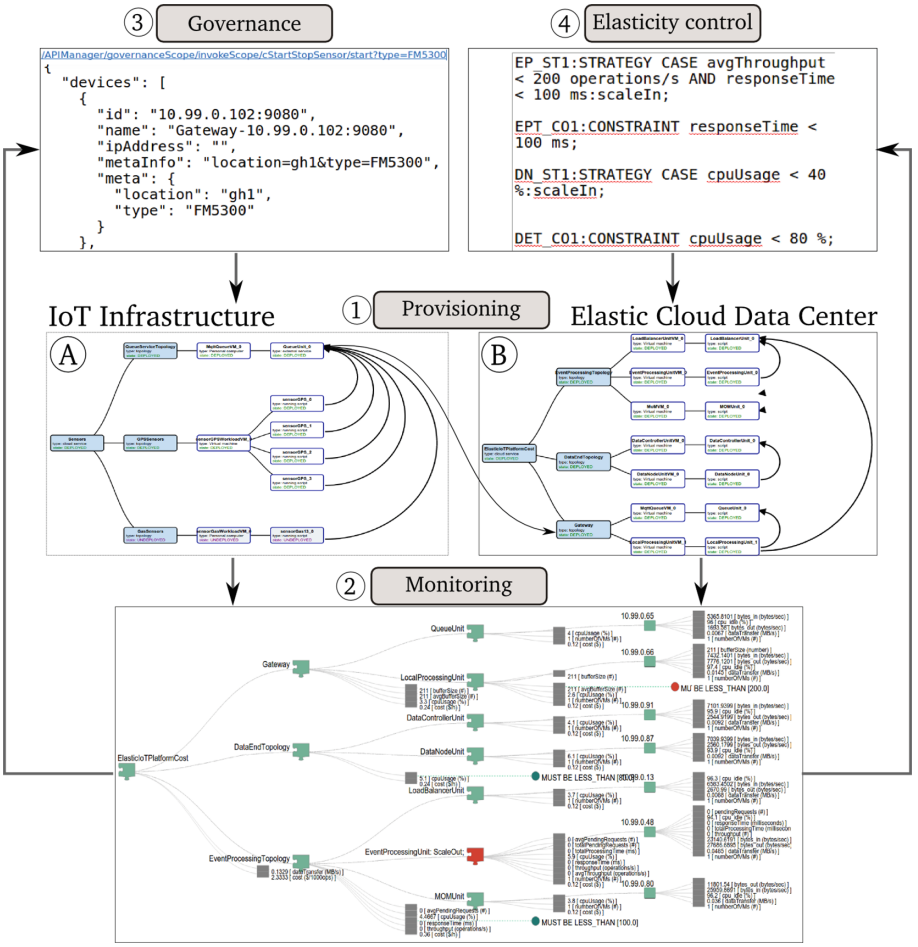

Fig. 2. Toolset for developing and testing elastic IoT cloud platforms

[1] See https://github.com/tuwiendsg/DaaSM2M and https://github.com/tuwiendsg/SDM.

5.2 Development and Deployment

First all of all, to make the (re)configuration of the IoT cloud platform flexible, using our toolset (Fig. 2), the predictive maintenance company can deploy two separate configurations: a configuration of the IoT sensors and gateways (Part Ⓐ), and another of the cloud data center (Part Ⓑ). This enables them to play with different sensors easily regarding data and topology, communication protocols, bursting workload, while the cloud services might be stable. For both configurations, IoT units and cloud services are provided from different providers from various repositories. The configurations will be specified in TOSCA and we *Provision* (①) them. Figure 2 – Part Ⓐ – shows an example of TOSCA-based sensor topology, which allows the developer to manage single sensors and sensor topologies[2] (and VM which hosts sensors in emulated scenarios). Figure 2 – Part Ⓑ – shows the deployment of an elastic IoT configuration platform – named `ElasticIoTPlatform` – at the data center to reflect real cloud services and emulated gateways. With our techniques, such configurations (for sensors and for gateways/services) can be also programmed using Java code, enabling different ways to program and test IoT cloud platforms.

Having the entire IoT cloud platform is provisioned, the company focuses on *Monitoring*(①). Before provisioning, platform developers must have in mind what monitoring data is relevant for the elasticity of the platform, and implement the necessary monitoring capabilities. A crucial factor in elastic platforms is that instances of units tend to appear/disappear dynamically at run-time as a result of scaling actions being enforced due to various elasticity requirements (e.g., platform performance, quality, cost). Thus, the company wants to avoid monitoring information being lost due to scaling in/out of individual units, and also to have and overview over the overall behavior of the platform units and not only individual unit instances. Thus, the platform developer must use our tool for deciding the contribution of a unit instance to the overall behavior of the entire platform, or individual units, and structure monitoring information according to the architecture of the platform. For example, the developer could decide that CPU usage of all unit instances must be averaged, and that the network data transfer must be summed.

After having the platform deployed and monitored, the company focuses on the various *Governance*(③) processes which must be enforced over the IoT sensors and gateways, arising from the company's different security, geo-political, or performance objectives. For example, an abnormal event might be detected by the IoT platform, such as dangerous gas detected in a smart building. In such a case, for better analyzing the cause of the event, the frequency and data collected might need to be changed. For enabling such dynamic changes, we can invoke sensor and gateway capabilities through their APIs for changing data collection frequency, or execute a complex process for changing the security levels and protocols used to send data. Leveraging these capabilities, we can enable processes for governing the gateways and sensors in different situations.

[2] See https://github.com/tuwiendsg/DaaSM2M/tree/master/Configurations/sensors for samples of TOSCA.

Governance processes might change the frequency, size, and mechanisms in which sensor data is collected, processed and sent to the cloud data center. Thus, an *Elasticity control*(④) mechanism is crucial for ensuring the performance and quality of the overall IoT cloud platform, especially during and after the execution of governance processes, through elasticity. To enable elasticity control, the platform developers must design and develop elasticity capabilities for the individual platform units, w.r.t, their type and purpose. Any capability that enables dynamic reconfiguration of any aspect or property of the platform units qualifies as an elasticity capability, and must be designed and implemented in the platform units, and enforced at run-time. For example, if a governance process increases the data collection frequency, the elasticity control mechanism should scale the cloud services in the platform to handle the load increase.

One lesson learned is that from architectural design, development and operation, we need to decide if all of these complex services, gateways and sensors should be specified and deployed in a single software configuration or not. It is possible but it is not flexible and it is hard to manage. On the other hand, from an IoT cloud platform provider perspective, it is typical to provide a platform that includes gateways (at the edge) connecting to cloud services (in the data center) and let the customer to deploy possible sensors and configure these gateways and services to fit into the customer need.

5.3 Elasticity Analytics and Control

After developing the `ElasticIoTPlatform` configuration, the developer can use our toolset for deploying and running it. At runtime, the developer is able to follow the behavior of the application using our monitoring features, in order to refine the elasticity and governance requirements and respectively policies. For such a complex use-case, which encompasses both IoT and cloud environments, there are two main control perspectives: (i) controlling the services deployed in the cloud which manage data processing and storing, (ii) controlling the IoT parts for addressing the governance policies.

In a critical situation, the entire `ElasticIoTPlatform` needs to react in order to localize or to better analyze the cause of the critical situation. For this, further data needs to be collected, for avoiding errors and miss-predictions. Figure 3 shows a process described by the developer for addressing such case, in which the `sensor push rate` is increased (i.e., due to governance policy), and the cloud services are allowed to scale to higher cost levels. The latter is intended to address the issue of cost limit in the elasticity requirements, as normally the developer specifies a cloud service cost limit, for urgency reasons. In a day-to-day case, with an increasing workload the cloud service would employ more and more virtual resources up to the cost limit, while in the critical situation, the cloud service can exceed the respective limit. From our experiences, we learned that, in our architectural designs, controlling elasticity of cloud software services should give sufficient powers to developers (e.g., controlling multiple software services at a time, different software stacks, both system and application level metrics), while maintaining a simple mechanism of elasticity control specification. Moreover, this

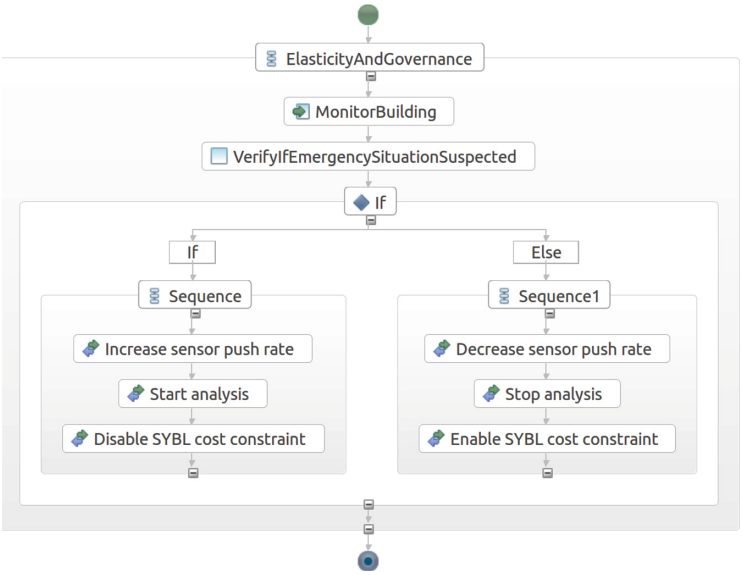

Fig. 3. Elasticity and governance process for the `ElasticIoTPlatform` configuration

control of gateways or of sensors, should interface with a variety of tools (e.g., different cloud providers, using different protocols, different gateway vendor-specific tools), for providing an end-to-end control of IoT cloud platforms.

5.4 Deployment and Failures

Let us consider some aspects related to the use of tools to evaluate IoT cloud platform deployment. We use our private DSG OpenStack, Stratuslab LAL public cloud (http://www.stratuslab.eu) and Flexiant public cloud. We run our deployment engine in our private OpenStack with m1.medium VM (2 CPU and 3,750 MB RAM) in the DSG cloud in order to test deployment issues for sensors and a configuration of an elastic IoT cloud platform – `ElasticIotPlatform`. While we deploy `ElasticIoTPlatform` in our DSG cloud, we want to emulate several sensors by deploying them in both clouds where on each m1.small VM (1 CPU, 2 GB RAM)[3], we deploy 30 sensors. We tested our studied configuration of an IoT cloud platform by deploying and activating from 100 to 350 sensors (when we use both clouds we deploy sensors equally in each cloud). Figure 4 show an increasing and varying trend of deployment failure rates. We can see that Flexiant has higher software failure rate by looking the deviation of failure percent of sensors and VMs, and VM failures are caused by the high number of concurrent requests on clouds. Our examples here are *not* about the performance issue of

[3] A CPU on DSG is T7700 @2.40 GHz, on Stratus is a QEMU CPU @2.20 GHz, and on Flexiant is QEMU CPU @3.10 GHz.

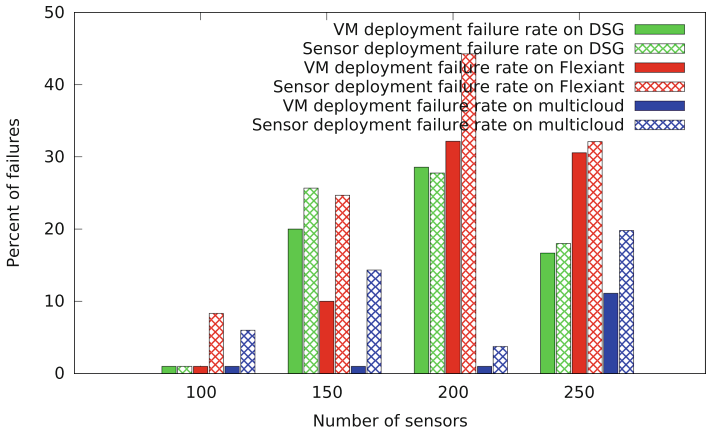

Fig. 4. Deployment failure rate for testing sensors in a multicloud environment

underlying clouds or deployment tools but show that by using a rich toolset, one can understand uncertain performance of clouds to utilize elasticity control and analytics features not only to deal with these performance and failure issues but to coordinate actions carried out at the IoT side.

6 Related Work

Several challenges of IoT and cloud integration are discussed intensively [5,7]. Many IoT platforms have been developed [2,21] based on which different services can be added. Our work is not about developing a particular IoT cloud platform, but focusing on techniques accelerating the development of such platforms. Although experiences have been shared, we have not seen similar experiences discussing rapid end-to-end development of elastic IoT cloud platforms.

Several frameworks support the development of IoT, such as [8,10,17]. Industrial tools, such as Predix (http://www.predix.io) and Microsoft Azure IoT (https://azure.microsoft.com/en-us/documentation/suites/iot-suite), also allow us to write IoT sensors and connect the sensors to cloud services. But they do not support elasticity controls. In our work, we do not focus on programming IoT sensors but recombine existing units and deploy them cross-issue spanning both IoT and clouds. The IBM experimental Internet of Thing Workbench (console.ng.bluemix.net/catalog/services/internet-of-things-workbench) offers capabilities to design and simulate end-to-end IoT cloud systems, but it does not support end-to-end monitoring and elasticity control.

In [11] experiences and evaluations of cloud application portability have been provided. Such evaluations are useful for us to decide the infrastructure used for the cloud service part of the IoT PaaS. However, they have not focused on IoT clouds in general. Zarko et al. [22] describe the CUPUS middleware, part of the OpenIoT platform [23], which provides a functionality for dynamically

adding/removing sensors to/from an IoT platform spanning mobile networks and cloud infrastructures. We do not focus on particular platforms but we enable such a functionality. Mazhelis et al. [13] conduct a comparative study on existing IoT platforms. The authors emphasize the need of having complex IoT platforms and supporting design and implementation phase and operation phase, although none of the compared ones (e.g., Xively, Axeda, Etherios) fully supports the end-to-end requirements. Our work aims at providing tools for such need.

There are some approaches on supporting simulation of IoT and IoT cloud systems, such as [24]. However, they are purely simulation systems, while we support configuration and testing of emulated sensors and gateways running in the cloud that interact with real-world cloud systems.

7 Conclusions and Future Work

In this paper, we described requirements, toolsets and engineering analytics for elastic IoT cloud platforms that simplify and accelerate the development of IoT cloud platforms, based on our development of the iCOMOT. Given the complexity of IoT cloud platform development requirements, it is hard to find any single, even powerful, toolset that will meet all the requirements. Therefore, we have to carry out appropriate engineering actions and also integrating different tools into our iCOMOT toolset. We show how utilizing such an integrated toolset we can simplify the development and and testing of IoT cloud platforms.

Currently, we focus on building a common knowledge of components, topologies and artifacts for supporting testing and evaluation of uncertainties in elastic IoT cloud platforms, in particular, and cyber-physical systems, in general.

References

1. Crowd analytics archives. http://www.dfrc.ch/tag/crowd-analytics/
2. The pacific controls galaxy. http://pacificcontrols.net/products/galaxy.html. Accessed 17 May 2016
3. U-test geo sports case study. http://www.u-test.eu/use-cases/#tab-1429727705-1-5
4. Akpinar, K., Hua, K.A., Li, K.: ThingStore: a platform for internet-of-things application development and deployment. In: Eliassen, F., Vitenberg, R. (eds.) Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, DEBS 2015, Oslo, Norway, 29 June - 3 July, 2015, pp. 162–173. ACM (2015)
5. Alamri, A., Ansari, W.S., Hassan, M.M., Hossain, M.S., Alelaiwi, A., Hossain, M.A.: A survey on sensor-cloud: architecture, applications, and approaches. IJDSN 2013 (2013). http://dx.doi.org/10.1155/2013/917923
6. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the 1st Edition of the MCC Workshop on Mobile Cloud Computing. MCC 2012, pp. 13–16. ACM, New York (2012)
7. Botta, A., de Donato, W., Persico, V., Pescape, A.: On the integration of cloud computing and internet of things. In: 2014 International Conference on Future Internet of Things and Cloud (FiCloud), pp. 23–30, August 2014

8. Chauhan, S., Patel, P., Sureka, A., Delicato, F.C., Chaudhary, S.: IoTSuite: a framework to design, implement, and deploy IoT applications: demonstration abstract. In: Proceedings of the 15th International Conference on Information Processing in Sensor Networks, pp. 37:1–37:2. IEEE Press, Piscataway (2016)
9. He, W., Yan, G., Xu, L.D.: Developing vehicular data cloud services in the IoT environment. IEEE Trans. Ind. Inf. **10**(2), 1587–1595 (2014)
10. Hong, K., Lillethun, D., Ramachandran, U., Ottenwälder, B., Koldehofe, B.: Mobile fog: a programming model for large-scale applications on the internet of things. In: Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing. MCC 2013, pp. 15–20. ACM, New York (2013)
11. Katsaros, G., Menzel, M., Lenk, A., Rake-Revelant, J., Skipp, R., Eberhardt, J.: Cloud application portability with TOSCA, chef and openstack. In: 2014 IEEE International Conference on Cloud Engineering, Boston, MA, USA, 11–14 March 2014, pp. 295–302. IEEE (2014). http://dx.doi.org/10.1109/IC2E.2014.27
12. Li, T., Liu, Y., Tian, Y., Shen, S., Mao, W.: A storage solution for massive IoT data based on NoSQL. In: GreenCom, pp. 50–57. IEEE (2012)
13. Mazhelis, O., Tyrvainen, P.: A framework for evaluating internet-of-things platforms: application provider viewpoint. In: 2014 IEEE World Forum on Internet of Things (WF-IoT), pp. 147–152, March 2014
14. Nastic, S., Sehic, S., Le, D.H., Truong, H.L., Dustdar, S.: Provisioning software-defined IoT systems in the cloud. In: FiCloud (2014)
15. Pereira, P.P., Eliasson, J., Kyusakov, R., Delsing, J., Raayatinezhad, A., Johansson, M.: Enabling cloud connectivity for mobile internet of things applications. In: Proceedings of the 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering. SOSE 2013, pp. 518–526 (2013). http://dx.doi.org/10.1109/SOSE.2013.33
16. Petrolo, R., Loscrí, V., Mitton, N.: Towards a smart city based on cloud of things. In: Proceedings of the 2014 ACM International Workshop on Wireless and Mobile Technologies for Smart Cities. WiMobCity 2014, pp. 61–66. ACM, New York (2014)
17. Riliskis, L., Hong, J., Levis, P.: Ravel: programming IoT applications as distributed models, views, and controllers. In: Proceedings of the 2015 International Workshop on Internet of Things Towards Applications. IoT-App. 2015, pp. 1–6. ACM, New York (2015). http://doi.acm.org/10.1145/2820975.2820977
18. Satyanarayanan, M., Lewis, G.A., Morris, E.J., Simanta, S., Boleng, J., Ha, K.: The role of cloudlets in hostile environments. IEEE Pervasive Comput. **12**(4), 40–49 (2013). http://dx.doi.org/10.1109/MPRV.2013.77
19. Truong, H.L., Copil, G., Dustdar, S., Le, D., Moldovan, D., Nastic, S.: ICOMOT - a toolset for managing iot cloud systems. In: 16th IEEE International Conference on Mobile Data Management, MDM 2015, Pittsburgh, PA, USA, 15–18 June 2015, vol. 1, pp. 299–302. IEEE Computer Society (2015)
20. Truong, H.L., Dustdar, S.: Principles for engineering IoT cloud systems. IEEE Cloud Comput. **2**(2), 68–76 (2015)
21. Xively. https://xively.com/. Accessed 17 May 2016
22. Zarko, I., Pripuzic, K., Serrano, M., Hauswirth, M.: IoT data management methods and optimisation algorithms for mobile publish/subscribe services in cloud environments. In: 2014 European Conference on Networks and Communications (EuCNC), pp. 1–5, June 2014
23. Zaslavsky, A.B., Perera, C., Georgakopoulos, D.: Sensing as a service and big data. CoRR abs/1301.0159 (2013). http://arxiv.org/abs/1301.0159
24. Zeng, X., Garg, S.K., Strazdins, P.E., Jayaraman, P.P., Georgakopoulos, D., Ranjan, R.: IOTSim: a cloud based simulator for analysing iot applications. CoRR abs/1602.06488 (2016). http://arxiv.org/abs/1602.06488