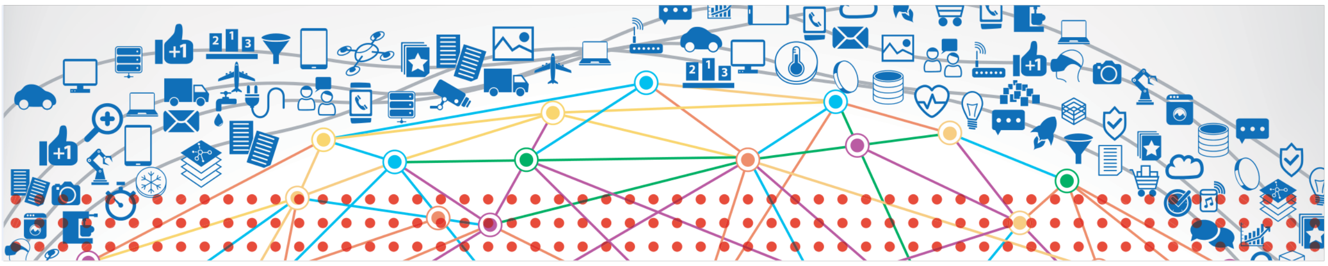

Principles for Engineering IoT Cloud Systems

Hong-Linh Truong and Schahram Dustdar, Vienna University of Technology

This invited article explores how Internet of Things (IoT) cloud systems could provide a coherent software layer for continuous deployment, provisioning, and execution of applications for various domains.

Recently, we've seen a wide adoption and deployment of Internet of Things (IoT) infrastructures and systems for various crucial applications,[1] such as logistics, smart cities,[2] and healthcare. This has led to high demands on data storage, processing, and management services in cloud-based datacenters, engendering strong integration needs between IoT and cloud services. Cloud services are mature and provide excellent elastic computation and data management capabilities for IoT. In addition, as IoT systems become complex, cloud management techniques are increasingly employed to manage IoT components. Thus, cloud services now act as computational and data processing platforms as well as management platforms for IoT. From a high-level view, IoT appears to be well-integrated with cloud datacenters to establish a uniform infrastructure for IoT cloud applications. However, the software layers on top of such integrated infrastructures are still fragmented, and therefore far from a uniform software layer to support a coherent execution environment for complex applications.

IoT elements (sensors, actuators, gateways, lightweight applications, and so on) are developed, deployed, and operated separately from cloud services (such as storage and data processing). Because of the complexity of software ecosystems, IoT providers are increasingly different from cloud providers with regard to communication protocols, software layers, and provisioning models, to name just a few. In addition, although cloud services, such as load balancers, message-oriented middleware, NoSQL storage, and streaming data processing frameworks, are designed to accept workloads and data from IoT, they lack capabilities to be coordinated with IoT operations. For example, most cloud services reactively monitor the load from IoT and adjust their performance behavior, but rarely communicate back to the IoT elements to steer the load generated by the IoT.

There are various reasons for these issues. Cloud services and IoT are created separately by cloud providers and IoT providers. In addition, the complexity of the IoT and cloud ecosystems prevents a single stakeholder from offering software that works well in IoT cloud systems. These issues prevent us from developing and operating IoT cloud systems in a co-

2325-6095/15/$31.00 © 2015 IEEE

IoT CLOUD SYSTEMS: SOME ENGINEERING PRINCIPLES

There are various ways to integrate Internet of Things (IoT) elements (such as sensors, actuators, and gateways) and cloud services in datacenters,[1] in which several "things" are connected to these services using software and layered protocols.[2,3] Several IoT platforms, such as Pacific Controls Galaxy (http://pacificcontrols.net/products/galaxy.html) and xively (https://xively.com), have also been developed.[4] In these IoT platforms, sensors and actuators are connected to and/or accessible from datacenters via gateways or intermediate nodes.

Ivana Podnar Zarko and her colleagues describe a middleware for dynamically adding and removing sensors to and from an IoT platform spanning mobile networks and cloud infrastructures.[5] In general, most work supports the development of either the IoT elements or the cloud services for IoT; uniform IoT cloud systems blending IoT and cloud services to provide a coherent execution environment for complex applications are still far from mature.

Although several principles have been studied for cloud computing elasticity,[6] cloud software engineering,[7] and cyber-physical systems,[8] we haven't seen such principles for engineering software layers on top of IoT cloud systems that offer a uniform view on development, deployment, provisioning, and operation of IoT cloud applications.

References

1. R. Petrolo, V. Loscrí, and N. Mitton, "Towards a Smart City Based on Cloud of Things," *Proc. ACM Int'l Workshop Wireless and Mobile Technologies for Smart Cities* (WiMobCity 14), 2014, pp. 61–66.
2. P.P. Pereira et al., "Enabling Cloud Connectivity for Mobile Internet of Things Applications," *Proc. IEEE 7th Int'l Symp. Service-Oriented System Eng.* (SOSE 13), 2013, pp. 518–526.
3. F. Bonomi et al., "Fog Computing and Its Role in the Internet of Things," *Proc. 1st Edition of the MCC Workshop on Mobile Cloud Computing* (MCC 12), 2012, pp. 13–16.
4. H.-L. Truong and S. Dustdar, "Sustainability Data and Analytics in Cloud-Based M2M Systems," *Big Data and Internet of Things: A Roadmap for Smart Environments*, N. Bessis and C. Dobre, eds., Studies in Computational Intelligence, vol. 546, Springer, 2014, pp. 343–365.
5. I. Zarko et al., "IoT Data Management Methods and Optimization Algorithms for Mobile Publish/Subscribe Services in Cloud Environments," *Proc. European Conf. Networks and Comm.* (EuCNC 14), 2014, pp. 1–5.
6. S. Dustdar et al., "Principles of Elastic Processes," *IEEE Internet Computing,* vol. 15, no. 5, 2011, pp. 66–71.
7. V.D. Cunsolo et al., "Applying Software Engineering Principles for Designing Cloud@Home," *Proc. 10th IEEE/ACM Int'l Conf. Cluster, Cloud, and Grid Computing* (CCGrid 10), 2010, pp. 618–624.
8. M. Broy and A. Schmidt, "Challenges in Engineering Cyber-Physical Systems," *Computer,* vol. 47, no. 2, 2014, pp. 70–72.

herent manner on top of infrastructures that blend various types of resources. Thus, it's hard to control and manage both IoT and cloud services as a uniform software layer (see the "IoT Cloud Systems: Some Engineering Principles" sidebar for further discussion).

However, further integration between IoT and cloud services as well as emerging complex applications require a uniform software layer view on top of these blended IoT elements and cloud services. A tight integration between IoT and cloud services allows coordination among IoT and cloud services. That is, a cloud service could ask an IoT service, which includes several IoT elements, to reduce the amount of sensing data or the IoT service could ask cloud services to prepare more resources for future incoming data. Consider the example of a sensor data as a service consisting of several hundred sensors monitoring building chillers.[3] We deploy such a

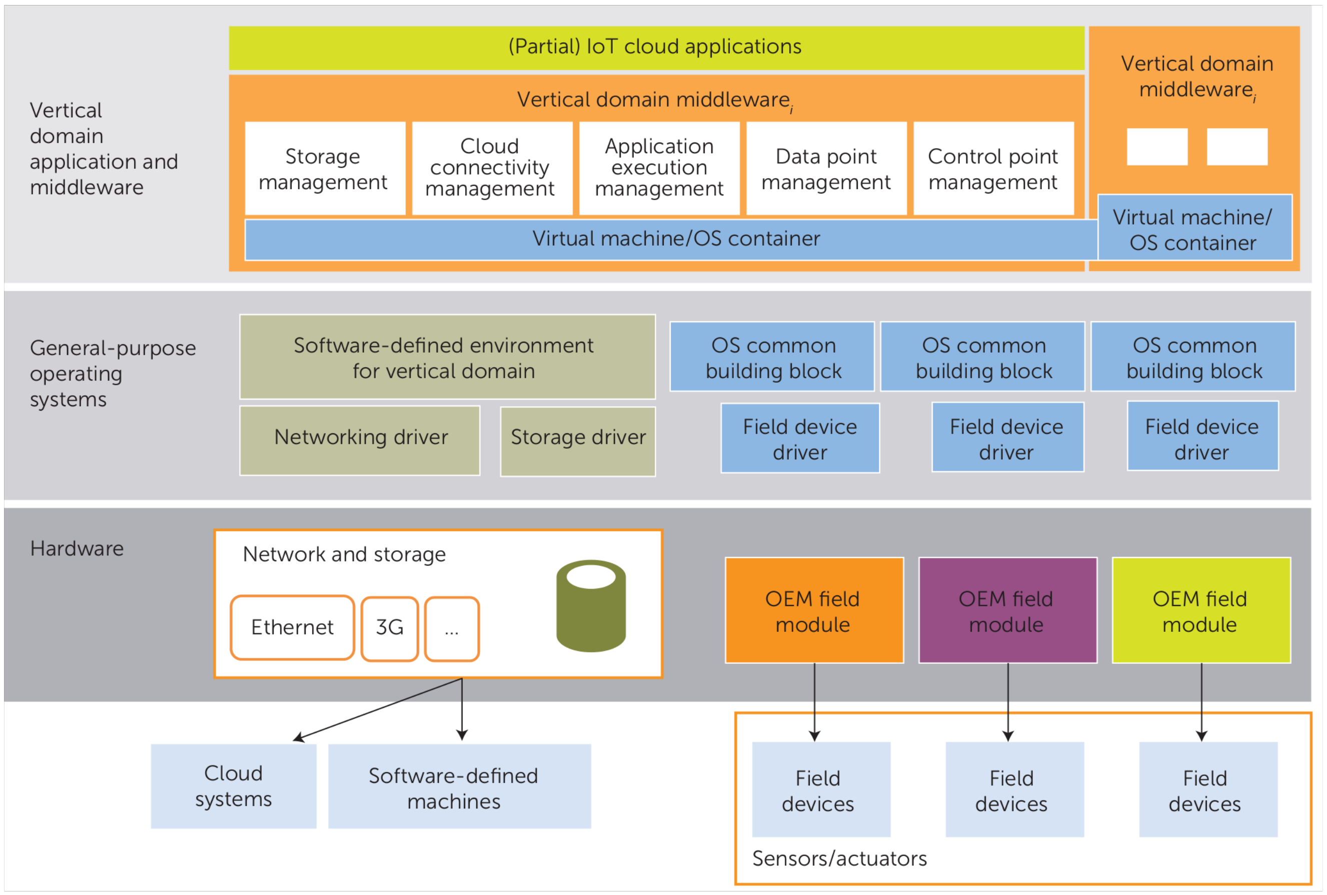

FIGURE 1. Software-defined machine (SDM) for the Internet of Things (IoT) part, which includes infrastructures and IoT elements deployed next to the "things" (at the edge of but not in the datacenter).

service in the IoT part, which includes infrastructures and IoT elements deployed next to the "things" (at the edge of but not in the datacenter), by preconfiguring the sensors' data read rates and activating only a subset of sensors. The activated sensors send monitoring data to the cloud services, which process the streaming data to check chiller behaviors. Such data analytics might provide adequate outcome to signal some abnormal behavior. To achieve high-accuracy analytics, from the cloud side we could activate more sensors and increase the read rates of sensors on the fly. But, if this action isn't coordinated with the control of cloud services, these services might not react quickly to deal with a sudden voluminous amount of incoming data. For example, it might take several minutes to acquire and instantiate new computing resources for cloud services; in other words, elasticity support in clouds might not meet our needs.[4] From this perspective, we need a different way to program, control, and manage both IoT and cloud services in a dynamic and flexible way.

Lifting the IoT Software Stack for IoT Cloud Systems

To date, certain parts of the IoT cloud system, such as networks and cloud services, can be fairly well controlled, but IoT elements can't. Analogous to cloud services in datacenters, which we can easily select, combine, and deploy to provide a platform suitable for a particular application or domain, the software stack for IoT services should support virtualization and composition. The key principle we want to support here is continuous, end-to-end engineering and elasticity. To lift the IoT software stack to a suitable abstraction, we could build a software-defined machine (SDM) by composing different IoT units. In this context, a SDM is similar to a small cloud infrastructure in which we deploy different IoT units and perform dynamic configuration and control of these units at runtime.

As Figure 1 shows, the SDM for IoT has three layers of software and hardware. At the bottom of the SDM are flexible and rich configurations of hardware to support (cloud) networking, enabling

IoT services to switch between communication hardware devices that support software-defined networking (SDN) for different loads. The storage should allow us to plug in different storage devices. Finally, OEM field device modules let us easily integrate different field devices (such as sensors and actuators) that have different device interfaces and communication protocols.

In the middle layer, general-purpose operating systems with virtualization capabilities include both common building blocks for the operating system and new building blocks.

Network drivers interface to networking hardware and can be used to control and manage the cloud network at runtime. Network drivers together with the cloud networking hardware act like a software-defined router,[5] which can be programmed to control communication protocols, interfaces, and so on. The data routed from this hardware can go to another SDM or to cloud services.

Storage drivers and the underlying storage would allow us to store data in an SDM locally or directly to the cloud storage. The important issue here is that by storing data into the SDM storage using software-defined APIs, we could configure whether to store the data locally or forward it to another SDM or cloud service.

The software-defined environment for a virtual domain allows us to create and launch vertical domain middleware and its services and applications. A virtual domain middleware would include a lightweight virtual machine/operating system container plus many components customized to the application domain.

In the top layer, the focus of our work, the vertical domain middleware is a software system that includes several components running on top of a virtual machine or operating system container. It consists of five main building blocks:

- *Cloud connectivity management* provides APIs that the application uses to exploit features of SDN in the application logic. For example, the application can decide to switch from broadband to mobile communication within its application logic at runtime.
- *Storage management* lets applications store data locally or remotely in a simple, unified way. Configuration of where the data is stored should be done via software-defined APIs.
- The *application execution environment* runs and manages specific applications on top of an SDM. Applications can be deployed, executed, stopped, and so on at runtime based on software-defined APIs. Applications will exploit all other features of the vertical domain.
- *Datapoint management* lets applications create, merge, read, and write the datastreams they need. Datastreams are based on data from field device drivers. From a programming perspective, a datapoint is an abstraction with software-defined APIs to support datastream operations (such as reducing or increasing the data read rate).
- *Control point management* lets us exploit software-defined capabilities to control field devices and their connections to the SDM. For example, the application can use control point management to disconnect a field device from the SDM or to increase a field device's read rate at runtime.

From the middleware, several vertical domains can be deployed and provisioned in an SDM using a software-defined environment for the vertical domain. When an SDM is realized with virtualization and composition capabilities such as cloud services, we can focus on engineering issues for coherent IoT cloud systems.

Engineering Perspectives for Coherent IoT Cloud Systems

By lifting the IoT software stack (for example, using the SDM concept), we could glue various IoT and cloud infrastructures and software to create IoT cloud systems. Table 1 shows possible hardware, software services, and protocols we could use to build an IoT cloud system.

Although IoT cloud systems consist of different types of infrastructures, software developers and service providers would expect to see uniform engineering techniques because they would develop and operate their software in a uniform software layer. An IoT cloud system that enables a software layer across both cloud and IoT infrastructures for customers must support the following functions.

End-to-end engineering and optimization. Developers must be able to develop and optimize code in an end-to-end view, and providers must consider end-to-end properties. For example, when composing components to design an IoT cloud application, we should be able to select sensors, gateways, message-oriented middleware (MOM), load balancers, and so on. We also need to create suitable topologies of these components and deploy them in the same manner, although the components would be executed in different parts of the IoT cloud systems, and we'd need various underlying techniques to support

Table 1. Sample of infrastructure, protocols, and software platforms for establishing an Internet of Things (IoT) cloud system.

Types	IoT	Clouds	Purpose
Infrastructure machines	Industrial and common gateways (for example, Intel IoT Gateway) and operating system containers (such as Dockers)	Virtual machines and operating system containers	Enable (virtual) machines where software components will be executed
Connectivity protocols	Message Queue Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), HTTP, control area network (CAN) bus	MQTT, Advanced Message Queuing Protocol (AMQP), HTTP, and so on	Enable connectivity among IoT elements and between the IoT part and cloud services
Platform software services	Lightweight data services (such as NiagaraAX/Obix), lightweight complex event processing (CEP) and data fusion, topology description and deployment service (such as TOSCA), and lightweight application containers (such as OSGI and Sedona)	Load balancers (such as HAProxy), message-oriented middleware (MOM) (such as ActiveMQ and Kafka), NoSQL, stream/batch processing (such as Hadoop and Spark), component repositories/marketplaces, and deployment services (such as TOSCA, HEAT, and Chef)	Enable core platform services for IoT and cloud tasks

this. Even though we provision these components at different places, the operator of such applications would focus on end-to-end service objectives, rather than just local properties of particular components or parts of the applications in the cloud or in the IoT infrastructure.

Development and production symbiosis. IoT cloud systems are complex. After deploying such systems, we could change the configuration but a reconfiguration must not stop all running components in the system. Thus, we must develop, deploy, and operate IoT cloud systems while continuing to add and test emulated or simulated IoT parts or cloud services together with the running (production) IoT cloud systems. For this purpose, engineering techniques must support possibilities for deploying and connecting emulated and simulated IoT and cloud service parts in a running system.

Elasticity coherence. Resources blended into an IoT cloud system and runtime demands for the IoT cloud system are both highly dynamic. Principles of elasticity will be applied in different parts of the IoT cloud system, but the elasticity must be coordinated among IoT and cloud services to ensure coherence.[1] For example, when increasing the number of sensors (elasticity of sensors as data resources), we should similarly increase the computing resources for cloud services, which process and store sensor data.

Engineering Principles for IoT Cloud Systems

Engineering principles for IoT cloud systems should support the functions we've described throughout the different phases of the IoT cloud system lifecycle:

- The *development phase* includes techniques for selecting, composing, and integrating components across the IoT cloud system for specifying and developing possible governance and control operations.
- The *deployment and provisioning phase* includes techniques for deploying various types of software components in the IoT cloud system at different levels of abstractions; and capabilities to configure and connect deployments and allow continuous provisioning.
- The *operation phase* includes capabilities to monitor end-to-end metrics, perform governance processes across the systems, and control coordinated elasticity processes.

Next, we describe the seven main principles for engineering IoT cloud systems. Table 2 summarizes how these principles support phases in the system lifecycle.

Principle 1: Enable virtualization and composition of IoT components as units. This principle relates to the development phase. Because of the IoT's diversity and complexity, we need powerful abstractions to simplify the integration and configuration of IoT components. We must enable selection and composition of software elements for the IoT services from multiple providers, similar to the manner in which we do so for cloud services. Thus, we apply the mature concept of virtualization and service composition in the cloud to IoT infrastructures and platforms, enabling the applicability of virtualization and composition techniques and tools throughout the IoT cloud systems. This calls for new develop-

Table 2. Summary of highlighted principles for engineering IoT cloud systems.

Principle	Development phase	Deployment and provisioning phase	Operation phase	End-to-end engineering and optimization	Development and production symbiosis	Elasticity coherence
1	+	+	+	+	+	+
2	+			+	+	
3	+	+	+	+		
4	+	+	+	+	+	+
5	+		+	+		+
6	+		+	+		
7			+	+		+

Note: A "+" indicates that the principle supports the listed perspective and phase.

ment of the concept service units for the IoT, such as software-defined IoT units.[6] We need such units for abstracting sensor data points, actuator control points, cloud network connectivity, and IoT gateways. Virtualization of IoT components also enables elasticity and a pay-per-use model of diverse types of IoT units for different vertical domains.

Principle 2: Enable emulated/simulated IoT parts working with production cloud services. This principle is related to both the development and operation phases. Although we can develop and deploy cloud services in IoT cloud systems entirely in cloud software infrastructures, we can't do the same for IoT parts in many cases. During the development phase, it's difficult to deploy real sensors, actuators, and gateways in a large-scale setting to test if they'd work. Therefore, engineering tools must support emulated/simulated sensors, actuators, and gateways, fostering symbiotic development and operation engineering actions. On the one hand, we need to develop emulated/simulated sensors, actuators, and gateways. Such emulated/simulated elements are, in principle, real software, but they're deployed in emulated environments. For example, hundreds of sensors can be deployed in a virtual machine. However, these sensors should be able to produce realistic sensor data and be controlled from cloud services. The use of realistic data, high-level emulated sensors, and lightweight virtual machines would be a good starting point.

Principle 3: Enable dynamic provisioning of IoT and cloud service units through uniform marketplaces and repositories for multiple stakeholders. This principle is related to all phases of IoT cloud systems in which different stakeholders (developers, infrastructure providers, software providers, and end users) will develop, sell, and operate IoT and cloud units. Equipped with virtualization and composition tools, these stakeholders could develop and sell different types of units, which can be easily offered via mature data/software marketplaces and repositories.[7] IoT cloud system engineering tools should interface with existing dynamic marketplaces in an integrated and uniform manner to enable pay per use, compatibility checks, and dynamic software packaging for the provisioning processes.[8]

Principle 4: Provide multilevel software stack deployment and configuration. This principle is related to the deployment and provisioning and operation phases. With capabilities supported from principles 1–3, both the IoT cloud system and the applications will require on-demand deployment and configuration of diverse types of software units. Obviously, these units range from virtual machines, operating system containers, network connectivity components, and sensors, to lightweight application and heavyweight cloud services. Any deployment and configuration tool for the IoT cloud system must deal with different software stack layers. Contemporary deployment and configuration tools, on the other hand, mainly work with a single software stack, such as static deployment at the virtual machine level or dynamic deployment at the application container level. This principle forces us to combine different deployment and configuration techniques to provide a cross-software stack and a cross-infrastructure deployment.

Principle 5: Provide software-defined elasticity and primitive governance functions for all IoT and cloud service units. This principle is related to the development of IoT cloud systems and is also strongly related to other principles for the operation phase. Software-defined elasticity and primitive governance functions let us control units individually at runtime. Having these functions would let us develop complex governance and elasticity control processes by composing the primitive functions from different units. This will also let us support end-to-end coordinated elasticity control across the entire IoT cloud system.

Principle 6: Provide monitoring and analysis for an end-to-end view on elasticity and dependability properties. This principle is related to the operation and development phases. Clearly, various existing monitoring and analysis tools will allow us to monitor and understand different behaviors of IoT units, cloud services, and networks. But what's important to the developer and the operator is to have a clear end-to-end view of software behavior so they can optimize the development and operation. For end-to-end elasticity metrics that characterize the behavior of IoT cloud systems,[9] we need complex composable methods to evaluate them by leveraging different types of monitoring data vertically and horizontally correlated across the whole IoT cloud system. Thus, we must focus on connecting different types of monitoring systems and present new metrics characterizing the end-to-end behavior.

Principle 7: Coordinate elasticity to enable a coherent elastic execution throughout the whole IoT cloud system. This principle is related to the operation phase. Elasticity will be seen at different parts of the IoT cloud system. We could have typical elasticity with regard to resources and quality in the cloud services. But we can also carry out different types of elasticity at the IoT part, such as provisioning and activating many more sensors to increase the quality of sensing data for data analytics. However, we need a closer loop among different parts of the IoT cloud system to optimize system operation. Therefore, we must support elasticity at various parts in a coherent way—for example, an elasticity strategy in the IoT part should be communicated to the cloud service part to prevent unexpected behaviors. One possibility is to examine the relationship between coordination and elasticity.[10] Another possibility is to extend current elasticity control algorithms in the cloud for IoT[11] by combining IoT governance processes.[12] Furthermore, we can also learn concepts for resource elasticity in federated clouds.[13,14] However, most of these techniques must be reworked to support end-to-end metrics. From the system perspective, IoT units and cloud services must support primitive actions so one can perform elasticity actions at runtime. This is highly related to issues in deployment, provisioning, and runtime governance.

Example Implementations

The principles we've described are based on our research and experience in developing various tools for IoT cloud systems. Two implementations demonstrate how we can apply these principles in software frameworks.

Software-Defined Machines

Principle 1 (on virtualization and composition of IoT components) is crucial to enabling the three functions discussed earlier. However, supporting virtualization and composition at the IoT part is challenging. By designing SDM concepts and tools,[6,8,12] we aim to support principle 1 and other related principles to make sure that IoT components can be easily composed, deployed, and controlled. Engineering such SDMs and other components in their ecosystems for coherent IoT cloud systems is challenging. Fortunately, several parts of the system can be leveraged by utilizing cloud services for the datacenter part and for IoT marketplaces. Over the past two years, we've concentrated on four aspects of SDMs: tools for building IoT units, governance of IoT units, marketplaces and provisioning units, and dynamic data and control points for sensors and actuators. At the moment, prototypes of governance, provisioning and marketplaces, and dynamic data and control points are available under open source licenses.

iCOMOT Toolset

As noted earlier, during the development and operation of the software layer of IoT cloud systems, we need different tools to enable different goals, such as deployment, provisioning, control, and monitoring. Our prototype, iCOMOT (http://tuwiendsg.github.io/iCOMOT), supports the seven principles and includes the tools shown in the bottom layer of Figure 2[15]:

- Salsa (Setting and Launching Service Applications) enables multilevel software stack deployment across IoT and cloud infrastructures.[16]
- GovOps (Governance and Operations) supports runtime governance management for IoT units and gateways.
- Mela (Monitoring Elasticity) enables elasticity monitoring and analytics across IoT cloud systems.
- rSYBL (runtime Simple Yet Beautiful Language) supports elasticity controls by invoking different

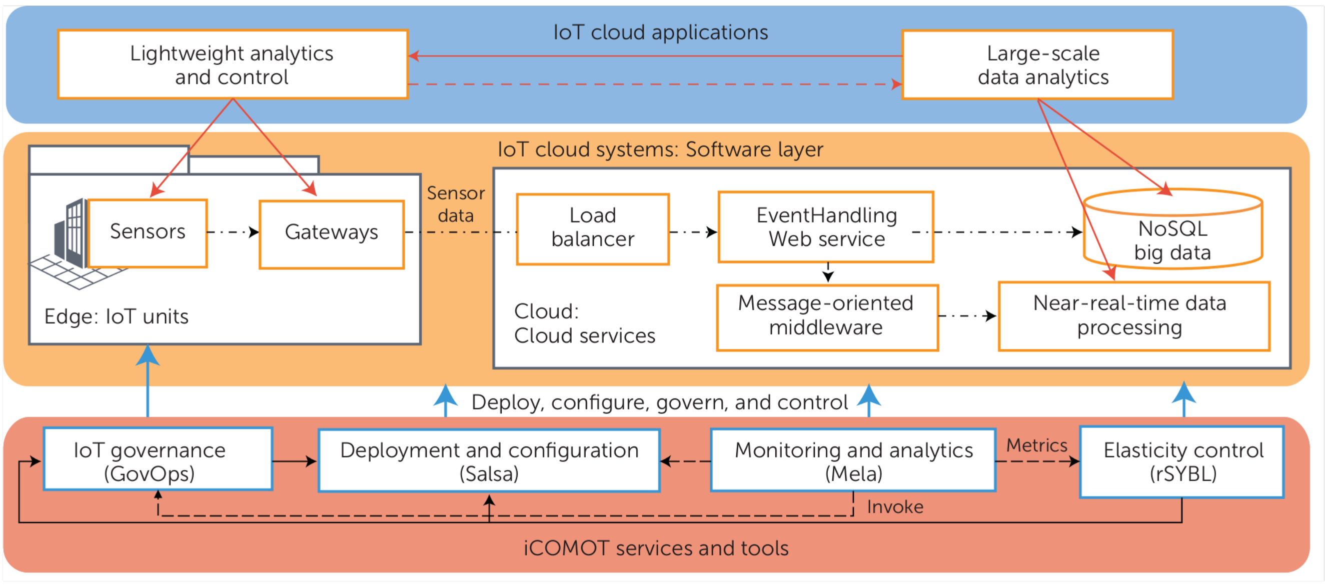
IoT cloud applications
Lightweight analytics and control
Large-scale data analytics
IoT cloud systems: Software layer
Sensors
Gateways
Sensor data
Edge: IoT units
Load balancer
EventHandling Web service
NoSQL big data
Cloud: Cloud services
Message-oriented middleware
Near-real-time data processing
Deploy, configure, govern, and control
IoT governance (GovOps)
Deployment and configuration (Salsa)
Monitoring and analytics (Mela)
Metrics
Elasticity control (rSYBL)
Invoke
iCOMOT services and tools

FIGURE 2. iCOMOT for IoT cloud systems.[15] The top layer represents typical IoT applications executed across IoT and clouds. The middle layer represents the software layer as an IoT cloud system built on top of various types of cloud services and IoT elements. The bottom layer shows different tools and services from iCOMOT that can be used to monitor, control, and configure the software layer.

strategies for cost, performance, and resources in a coordinated manner across IoT and cloud services.

Except for GovOps, which is designed for IoT units, many of iCOMOT's features are based on cloud service elasticity features extended for IoT units and SDMs.

As an example, consider the case of a predictive maintenance company that's interested only in managing sensors, actuators, and data processing and storage to support its maintenance analytics. The analytics includes lightweight analytics and control and data analytics, as shown in the top layer of Figure 2. The company requires a software layer as an IoT cloud system built on top of several rented IoT and cloud infrastructures, shown in the middle layer of Figure 2. Using deployment tools, iCOMOT could support the developer and the provider in joining different configurations of an IoT cloud system. For example, a sensor-data-as-a-service configuration could include several sensors to be deployed and controlled at runtime (activate, deactivate, change read rate, and so on). Such a configuration could be used to both deploy emulated sensors and control real sensors. This configuration could be connected to a configuration of real production cloud services and gateways. Using iCOMOT, one could perform some principles on dynamic elasticity configuration and control. Furthermore, sensors and services come from different providers and are hosted in different marketplaces or repositories. To deploy such very different configurations, our software must deal with multilevel software stack configuration and control.

Users can also observe and analyze high-level elasticity properties in an end-to-end view: from the entire IoT cloud system, to specific topologies of service units, to individual units. For example, costs for the software layer can be determined on the fly from costs for various gateways, data services, load balancers, and so on, based on their complex dependencies and sensor data rates. Such costs and data rates are fed to an elasticity controller that can enable flexible elasticity strategies at the IoT part or cloud services. For example, it can stop changing the sensor's read rate when the cost for the whole system violates user expectations.

Our current work has focused on the realization of the seven principles described here. As our initial results show, once we can provide such tools, we'll release the developer and provider from several obstacles to enable a smooth software layer on top of complex, hybrid, and blended IoT and cloud services. However, we haven't discussed other important principles related to security, privacy, and compliance in IoT cloud systems. We aim to study such principles in future work.

IEEE CLOUD COMPUTING Previous Page | Contents | Zoom in | Zoom out | Front Cover | Search Issue | Next Page Qmags THE WORLD'S NEWSSTAND®

Acknowledgments

For their fruitful discussions and contributions toward realizing some of the principles presented in this article, we thank our PhD students and colleagues, especially Georgiana Copil, Alessio Gambi, Christian Inzinger, Duc-Hung Le, Daniel Moldovan, Stefan Nastic, Sanjin Sehic, and Michael Vögler. The work mentioned in this article is partially supported by the European Commission through the CELAR FP7 project (FP7-ICT-2011-8 #317790), the H2020 LEIT U-test, and the Pacific Controls Cloud Computing Lab (pc3l.infosys.tuwien.ac.at).

References

1. A. Botta et al., “On the Integration of Cloud Computing and Internet of Things,” *Proc. Int’l Conf. Future Internet of Things and Cloud* (FiCloud 14), 2014, pp. 23–30.
2. J.M. Hernández-Muñoz et al., “Smart Cities at the Forefront of the Future Internet,” *The Future Internet*, J. Domingue et al., eds., Springer, 2011, pp. 447–462.
3. H.-L. Truong and S. Dustdar, “Programming Elasticity in the Cloud,” *Computer*, vol. 48, no. 3, 2015, pp. 87–90.
4. P.C. Brebner, “Is Your Cloud Elastic Enough? Performance Modelling the Elasticity of Infrastructure as a Service (IaaS) Cloud Applications,” *Proc. 3rd ACM/SPEC Int’l Conf. Performance Eng.* (ICPE 12), 2012, pp. 263–266.
5. K. Kirkpatrick, “Software-Defined Networking,” *Comm. ACM*, vol. 56, no. 9, 2013, pp. 16–19.
6. S. Nastic et al., “Provisioning Software-Defined IoT Cloud Systems,” *Proc. 2nd Int’l Conf. Future Internet of Things and Cloud* (FiCloud 14), 2014, pp. 288–295.
7. O. Krieger, P. McGachey, and A. Kanevsky, “Enabling a Marketplace of Clouds: VMware’s vCloud Director,” *Operating Systems Rev.*, vol. 44, no. 4, 2010, pp. 103–114.
8. M. Vögler et al., “LEONORE—Large-Scale Provisioning of Resource-Constrained IoT Deployments,” *Proc. 9th IEEE Int’l Symp. Service-Oriented System Eng.* (SOSE 15), 2015, pp. 78–87.
9. D. Moldovan et al., “MELA: Elasticity Analytics for Cloud Services,” *Int’l J. Big Data Intelligence*, vol. 2, no. 1, 2015, pp. 45–62.
10. S. Mariani et al., “Coordination-Aware Elasticity,” *Proc. 7th IEEE/ACM Int’l Conf. Utility and Cloud Computing*, 2014, pp. 465–472.
11. G. Copil et al., “On Controlling Cloud Services Elasticity in Heterogeneous Clouds,” *Proc. 7th IEEE/ACM Int’l Conf. Utility and Cloud Computing*, 2014, pp. 573–578.
12. S. Nastic et al., “rtGovOps: A Runtime Framework for Governance in Large-scale Software-defined IoT Cloud Systems,” *Proc. 3rd IEEE Int’l Conf. Mobile Cloud Computing, Services, and Eng.*, 2015, pp. 24–33.
13. F. Paraiso, P. Merle, and L. Seinturier, “Managing Elasticity across Multiple Cloud Providers,” *Proc. Int’l Workshop Multi-cloud Applications and Federated Clouds* (MultiCloud 13), 2013, pp. 53–60; doi:10.1145/2462326.2462338.
14. R.N. Calheiros et al., “A Coordinator for Scaling Elastic Applications across Multiple Clouds,” *Future Generation Computer Systems*, vol. 28, no. 8, 2012, pp. 1350–1362.
15. H.-L. Truong et al., “iCOMOT: Toolset for Managing IoT Cloud Systems,” demo, 16th IEEE Int’l Conf. Mobile Data Management, 2015.
16. D.-H. Le et al., “SALSA: A Framework for Dynamic Configuration of Cloud Services,” *Proc. 6th Int’l Conf. Cloud Computing Technology and Science*, 2014, pp. 146–153.

HONG-LINH TRUONG *is an assistant professor in the Distributed Systems Group at the Vienna University of Technology. His work focuses on service engineering analytics—in particular, for cloud computing; service-oriented architectures and computing; distributed and parallel computing; Internet of Things; complex and elastic distributed systems; and context-aware computing. Truong has a PhD and Habilitation in computer science from the Vienna University of Technology. Contact him at truong@dsg.tuwien.ac.at.*

SCHAHRAM DUSTDAR *is a full professor of computer science (informatics) and heads the Distributed Systems Group at the Vienna University of Technology. His work focuses on Internet technologies. Dustdar is a member of the Academy Europeana, an ACM Distinguished Scientist, and recipient of the IBM Faculty Award 2012. Contact him at dustdar@dsg.tuwien.ac.at.*

cn Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.

IEEE CLOUD COMPUTING Previous Page | Contents | Zoom in | Zoom out | Front Cover | Search Issue | Next Page Qmags THE WORLD'S NEWSSTAND®