Towards Automated IoT Application Deployment by a Cloud-based Approach

Fei Li, Michael Vögler, Markus Claeßens, Schahram Dustdar
Distributed Systems Group
Vienna University of Technology
Argentinierstrae 8/184-1, 1040 Vienna, Austria
Email: {lastname}@infosys.tuwien.ac.at

***Abstract*—Internet of Things solutions are typically domain-specific, relying on heterogeneous hardware, communication protocols and data models. In such system environments, the deployment of IoT applications is very intricate. The application environments differ from one system to another and service management procedures are non-standardized, making it hard for solution providers to efficiently deploy and configure applications for a large number of users. This paper proposes to employ TOSCA—a new standard for cloud service management—to systematically specify the components and configurations of IoT applications. We will demonstrate that, by using TOSCA, application models can be reused, and deployment processes can be automated in heterogeneous IoT system environments.**

I. Introduction

Internet of Things [1] solutions are typically domain-specific, relying on heterogeneous hardware (e.g. sensors, actuators and gateways), communication protocols and data models. To deal with such complexity, a lot of industrial and academic efforts are put into developing gateway frameworks that facilitate device integration and application development. However, such efforts have also led to many proprietary application runtime environments [2][3][4] with non-standardized service management processes.

In our previous work, we have developed the IoT PaaS [5] architecture to improve the efficiency and scalability of IoT service delivery. It allows IoT solutions to be delivered on a PaaS cloud as *virtual verticals*, which are composite, configurable, and able to share the underlying cloud platform services and computing resources with other IoT solutions. Although the architecture allows service providers to efficiently deliver and scale up IoT services, the service management tasks, such as application deployment, driver installation and gateway configuration are still handled manually in a case-by-case manner, due to the underlying heterogeneity of IoT infrastructures. The problem is that the state-of-the-art IoT service frameworks, either gateway-based or cloud-based, lack a systematic methodology to specify and maintain the intricate software and hardware dependencies in IoT applications.

This paper is motivated by the challenges we have experienced in deploying IoT solutions at various scales and in multiple application domains[1] (mostly building automation and vehicle tracking). In order to improve the reusability of service management processes and automate IoT application deployment in heterogeneous environments, we propose to employ Topology and Orchestration Specification for Cloud Applications (TOSCA) [6][7] for IoT service management. TOSCA is a new standard aiming at describing the topology of cloud applications by using a common set of vocabulary and syntax. In this paper, we will demonstrate the feasibility of using TOSCA to specify a typical IoT application in building automation—Air Handling Unit (AHU). The common IoT components such as gateways and drivers will be modelled, and the gateway-specific artifacts that are necessary for application deployment will also be specified. Based on the case-driven modeling, we will discuss our early experience gained from applying TOSCA for IoT applications. This work is in line with our ongoing effort of enabling the convergence of IoT and cloud [8]. To the best of our knowledge, this is also the first attempt of explicitly addressing the IoT application deployment problem using a cloud-based approach.

The rest of the paper is organized as follows: Section II will give an introduction to the background about TOSCA and other concepts in IoT frameworks. Section III will start to model the TOSCA nodes and relationships for AHU application. The gateway specific application artifacts are specified in Section IV. Then the experiences of using TOSCA are discussed in Section V. Section VI will present the related work and discuss how our work fits into the state of the art research on both IoT and TOSCA. The paper will be concluded in Section VII with future work.

II. Background

A. IoT PaaS

IoT services are often delivered in physically isolated verticals (often referred to as ”silos”), in which hardware, middleware and application logics are tightly coupled to fulfill domain or even project-specific requirements. IoT PaaS [5] is a novel IoT service delivery platform that leverages the service delivery model of PaaS cloud. On this architecture, we offer the possibility of providing end-to-end IoT solutions as *virtual verticals* on cloud, opposed to the traditional delivery model of physically-isolated and tightly-coupled vertical solutions. IoT PaaS is a generic, domain-independent architecture that relies on *domain mediators* to integrate domain-specific control protocols and data models. We have demonstrated the

[1] http://www.pacificcontrols.net/projects/ict-project.html

domain mediation mechanism with an oBIX (Open Building Information Exchange) [9] mediator for building automation applications. This paper further leverages this cloud platform with TOSCA to address the challenges in IoT service delivery.

B. Gateways

To handle the multitude of field devices in IoT solutions, gateways [3][4][10] are designed to connect heterogeneous, resource-constraint devices. Gateways support various device drivers and protocol stacks to communicate with devices, for example 6LoWPAN (IPv6 over Lower power Wireless Personal Area Networks). Depending on their applications, they may also support domain-specific, device-oriented data exchange protocols such as BACNet (Building Automation and Control Networks). Gateways can also provide service interfaces, such as the RESTful interfaces of oBIX and CoAP [11](Constrained Application Protocol) to ease the integration of lower-level IoT infrastructures with enterprise applications. In brief, the basic function of gateways is to provide an abstraction of IoT infrastructure by effectively translating device/network interfaces into software interfaces. The process is generally known as device virtualization [12]. On top of this core function, most modern gateways are also built with application runtime environments, which are usually proprietary or non-standardized.

C. TOSCA

The Topology and Orchestration Specification for Cloud Applications (TOSCA) is a new OASIS standard for improving portability of cloud applications in face of growingly heterogeneous cloud application environments. In the following we briefly describe the core concepts of TOSCA.

TOSCA specifies a meta-model for describing both the structure and management of IT services. The structure of a service is defined by the *Topology Template*, which consists of *Node Templates* and *Relationship Templates*. Together they represent a service by a directed graph. In this graph, every component is represented by a *Node Template* that instantiates a *Node Type*, which defines the properties and operations of a component. To support reusability, Node Types are defined separately and just referenced in Node Templates. Furthermore, in addition to the reference, usage constraints of components, e.g. number of occurrences, can be specified. In the topology of a service, nodes are connected by relations. *Relationship Templates* specify the relationship among nodes in the Topology Template, where each Relationship Template refers to a separately defined Relationship Type, which in turn defines the semantics and any properties that can be used to represent a relationship, such as ”dependOn” or ”connectTo”. The actual scripts, configuration files and application archives required by an application are called *Artifacts*, which are explicitly specified in *Artifact Types* and *Artifact Templates*. Artifacts are specific to each runtime environment and configuration of an application.

The management process of creating, deploying and terminating a service can be defined by *Plans*. Plans are process models that can be implemented as complex workflows. The specification of these process models relies on existing standards, such as BPMN or BPEL, to automate management processes in different application environments. TOSCA provides two ways of using plans—a container to use a reference of a process model (via *Plan Model Reference*) and to include an actual model in the plan (via *Plan Model*). The process model contains tasks that refer to operations of Interfaces of either Node Templates, Relationship Templates or any other available interface. This guarantees that a plan can directly manipulate nodes of the service topology or interact with external systems.

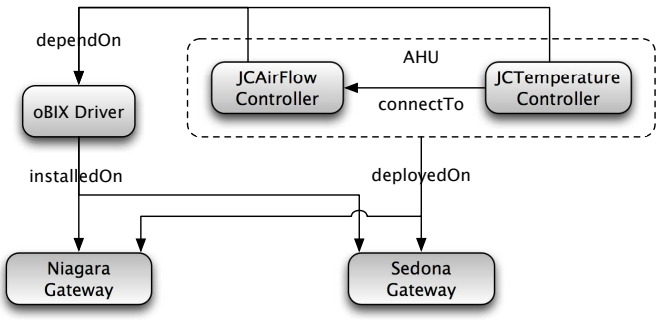

Fig. 1: Air Handling Unit usecase

The topology templates, plans and artifacts of an application are packaged in a Cloud Service Archive (.csar file) and deployed in a TOSCA environment, which is able to interpret the models and perform specified management operations.

It is worth noting that plans are not always required in using TOSCA. The TOSCA environment is able to infer the correct topology and management procedure just by interpreting the topology template. This is known as a ”declarative” approach. Plans realize an ”imperative” approach that explicitly specifies how each management process should be done. As the first attempt of employing TOSCA for IoT applications, this work uses the declarative approach, i.e. only applying the concepts in Topology Template.

III. Modeling IoT applications in TOSCA

This paper uses a typical IoT application in building automation systems to demonstrate the complexity of IoT applications as well as the feasibility of TOSCA in facilitating IoT application deployment.

A. Application description

Air Handling Unit (AHU) is a common facility in modern buildings. Its basic function is to condition and circulate air. In building automation solutions, sensors and actuators are applied to AHUs in order to remotely monitor and control them. Figure 1 illustrates a simplified deployment view of an AHU that is commonly found in commercial solutions[2]. Two core components produced by Johnson Controls (JC)—Air Temperature Controller and Flow Rate Controller—are connected to output fresh air at a temperature point set by an

[2]http://www.pacificcontrols.net/projects/ict-project.html

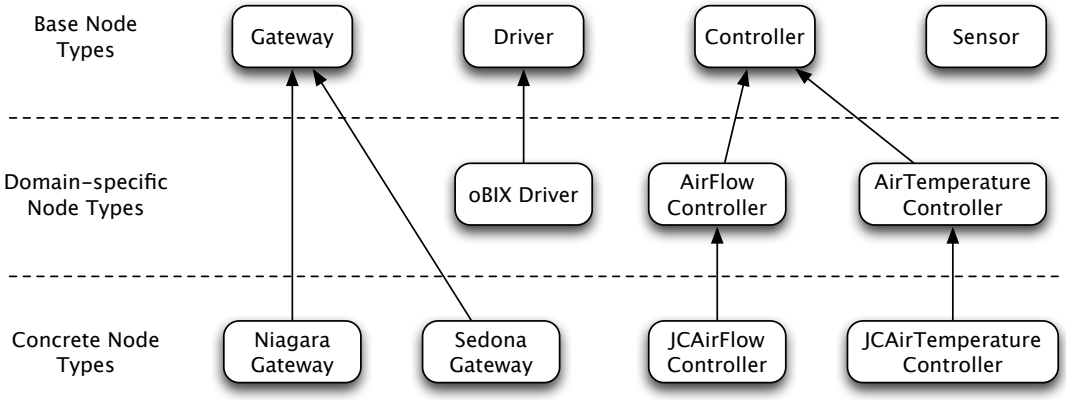

Fig. 2: Node types

operator. Other than the control interface defined by Johnson Controls, the AHU relies on the oBIX protocol for applications to access it. Such AHU applications will be deployed in various gateway models due to the technical requirements of other facilities (e.g., lighting) and legacy building automation systems. In this paper we demonstrate the deployment with two gateway frameworks—Niagara[3] and Sedona[4].

B. Modeling the nodes

Modeling nodes is the first step in using TOSCA to model IoT applications. Figure 2 illustrates the hierarchical node model we developed for the AHU application.

1) Base Node Types: The *Base Node Types* are directly derived from a generic TOSCA root node type. This puts them at the same level as other common cloud application nodes, including server, database and so on. The nodes at this level present the most fundamental concepts in IoT applications. Listing 1 presents the type definition of three basic node types, namely *Controller*, *Gateway* and *Driver*, in psydo-XML[5]. *Sensor* is not used in our application, thus not listed.

The most important element of the node types is the *Interface*. Interfaces define the operations that application providers can apply to the class of components. The operations presented in our example belong to a *Lifecycle interface*, which is able to instruct the TOSCA environment to change the status of these nodes. The concrete implementations of these node types need to provide corresponding interface implementations and define required parameters. The properties of these three node types are listed in Listing 2.

2) Domain-specific Node Types: The *Domain-specific Node Types* are related to IoT applications in a certain industrial domain, which is building automation in our case. For example, oBIX is a protocol widely used in building automation projects. It is based on web standards including XML, HTTP and URI to access building information and control facilities.

[3]http://www.niagaraax.com/cs/products/niagara_framework

[4]http://www.sedonadev.org

[5]For emphasizing the core concepts and saving space, we do not use name spaces. When the embedded structure of XML elements are too redundant, we also remove the end tags.

Listing 1: Examples of basic node types

```
<NodeType name="Controller">
  <DerivedFrom typeRef="RootNodeType" />
  <NodeTypeProperties element=
    "ControllerProperties" />
  <Interfaces>
    <Interface name="lifecycle">
      <Operation name="deploy" />
      <Operation name="configure" />
      <Operation name="start" />
      <Operation name="stop" />
      <Operation name="undeploy" />
  ...
</NodeType>

<NodeType name="Gateway">
  <DerivedFrom typeRef="RootNodeType" />
  <NodeTypeProperties element=
    "GatewayProperties" />
  <Interfaces>
    <Interface name="lifecycle">
      <Operation name="poweron" />
      <Operation name="poweroff" />
      <Operation name="reboot" />
  ...
</NodeType>

<NodeType name="Driver">
  <DerivedFrom typeRef="RootNodeType" />
  <NodeTypeProperties element=
    "DriverProperties" />
  <Interfaces>
    <Interface name="lifecycle">
      <Operation name="install" />
      <Operation name="uninstall" />
  ...
</NodeType>
```

AirFlowController and *AirTemperatureController* are common node types in AHU applications. Listing 3 demonstrates their description based on TOSCA.

Derived from the base controller properties, these two specific controllers add the controller-specific operations– *ChangeSetPoint* and *ChangeAirFlowRate*, respectively with *SetPoint* and *FlowRate* parameters. In our case, the controller properties are the same as the input parameters, thus not listed.

Listing 2: Properties of basic node types

```
<element name="ControllerProperties">
  <complexType>
    <sequence>
      <element name="Driver" type="string" />
  ...
</element>

<element name="GatewayProperties">
  <complexType>
    <sequence>
      <element name="User" type="string" />
      <element name="Password" type="string" />
  ...
</element>

<element name="DriverProperties">
  <complexType>
    <sequence>
      <element name="Version" type="string" />
  ...
</element>
```

Listing 3: Examples of domain-specific node types

```
<NodeType name="AirTempController">
  <DerivedFrom typeRef="Controller"/>
  <NodeTypeProperties element="AirTempProperties"/>
  <Interfaces>
    <Interface name="AirTempInterface">
      <Operation name="ChangeSetPoint">
        <InputParameters>
          <InputParameter name="SetPoint"
            type="xs:double"/>
  ...
</NodeType>

<NodeType name="AirFlowController">
  <DerivedFrom typeRef="Controller"/>
  <NodeTypeProperties element="AirFlowProperties"/>
  <Interfaces>
    <Interface name="AirFlowInterface">
      <Operation name="ChangeAirFlowRate">
        <InputParameters>
          <InputParameter name="FlowRate"
            type="xs:double"/>
  ...
</NodeType>
```

3) Concrete Node Types: The *Concrete Node Types* define the node types to be used in a specific application, with information about specific hardware and software vendors, models and versions. We only present the type definition of a Niagara Gateway in Listing 4 as an example, because other concrete node types follow the similar inherited relationship with their parent node as illustrated in Figure 2. The properties include information about the hardware model and software version.

Listing 4: Example of Concrete Node Types

```
<NodeType name="NiagaraGateway">
  <DerivedFrom typeRef="Gateway" />
  <NodeTypeProperties element=
    "NiagaraGatewayProperties"/>
</NodeType>
```

4) Node Templates: According to the TOSCA specification, *Node Templates* describe the specific instances of node types. The properties of a certain node type should be set in node templates. Essentially, Node Types describe the model of nodes, whereas Node Templates describe the actual nodes to be used in a certain application deployment. Listing 5 presents a template of Air Temperature Controller by Johnson Controls. Since there is an interface to set the output temperature, the *SetPoint* property can be changed at runtime.

Listing 5: Example of Node Templates

```
<NodeTemplate id="JCAirTempControllerTemp"
    name="Johnson Controls Air Temperature Controller"
    type="JCAirTempController">
  <Properties>
    <ControllerProperties>
      <Driver>oBIX</Driver>
    </ControllerProperties>
    <AirTempProperties>
      <SetPoint>21</SetPoint>
    </AirTempProperties>
  </Properties>
</NodeTemplate>
```

C. Modeling the relationships

The relationships required in our AHU application are common to many IoT applications. Listing 6 presents the four basic relationships illustrated in Figure 1. The names of the relationship types are self-explanatory. Each of the relationships are characterized by a source type and a target type. The relationship definitions can be used to specify the semantics of links between nodes and the methods of connections. Similar to node types, relationship types can also be inherited with more concrete properties, and instantiated into *Relationship Templates*. We will skip the definitions of templates and concrete properties due to limitation of space.

Listing 6: Relationship Types

```
<RelationshipType name="dependOn">
  <DerivedFrom typeRef="RootRelationshipType" />
  <ValidSource typeRef="Controller" />
  <ValidTarget typeRef="Driver" />
</RelationshipType>

<RelationshipType name="connectedTo">
  <DerivedFrom typeRef="RootRelationshipType" />
  <ValidSource typeRef="Controller" />
  <ValidTarget typeRef="Controller" />
</RelationshipType>

<RelationshipType name="installedOn">
  <DerivedFrom typeRef="RootRelationshipType" />
  <ValidSource typeRef="Driver" />
  <ValidTarget typeRef="Gateway" />
</RelationshipType>

<RelationshipType name="deployedOn">
  <DerivedFrom typeRef="RootRelationshipType" />
  <ValidSource typeRef="Controller" />
  <ValidTarget typeRef="Gateway" />
</RelationshipType>
```

IV. Artifacts in IoT application deployment

Artifacts are the actual scripts, files, packages, executables and all other necessary software pieces to be deployed in order to run an application. Common artifacts in cloud applications may include installation scripts, configuration files, archives and so on. For IoT applications, even though the basic artifact types are similar to cloud applications, the actual artifacts required by each application are highly dependent on the deployment environments. Based on the basic modelling in the previous section, we will demonstrate how TOSCA can help to manage IoT application deployment on heterogeneous environments.

A. Artifact Types

The two gateways used in our implementation—Niagara and Sedona—feature different runtime environments, programming languages and deployment procedures. However, the basic artifact types can be modeled in the same way as cloud applications[6].

1) File Artifact. Generic artifact type that contains certain information required during an application's lifecycle.
2) Script Artifact. Executable or interpretable artifact that encapsulates instructions in a script language for a certain operation.
3) Archive Artifact. A collection of files that are packaged for deployment.

For the deployment on gateway environments, we extend these basic artifact types to two other common types: *SourceArtifact* and *BinaryArtifact*. They are listed in Listing 7, and their properties in Listing 8. Sources are to be compiled by a compiler decided by the language of the source, whereas binaries are executed in a specific runtime environment.

Listing 7: Artifact types

```
<ArtifactType name="SourceArtifact">
  <DerivedFrom typeRef="FileArtifact" />
  <PropertiesDefinition element=
    "SourceArtifactProperties" />
</ArtifactType>

<ArtifactType name="BinaryArtifact">
  <DerivedFrom typeRef="FileArtifact" />
  <PropertiesDefinition element=
    "BinaryArtifactProperties" />
</ArtifactType>
```

B. Artifact Templates

Application deployment usually constitutes a series of operations specified by the vendor of a gateway. These operations transform artifacts, put them into specified locations, and set their status. We will model the Niagara and Sedona artifacts respectively in the following.

[6]http://docs.oasis-open.org/tosca/tosca-primer/v1.0/tosca-primer-v1.0.html

Listing 8: Properties of artifact types

```
<element name="SourceArtifactProperties">
  <complexType>
    <sequence>
      <element name="Language" type="string" />
      <element name="Compiler" type="string" />
  ...
</element>
<element name="BinaryArtifactProperties">
  <complexType>
    <sequence>
      <element name="Environment" type="string" />
  ...
</element>
```

1) Niagara artifacts: Similar to nodes and relationships, the actual artifacts used in an application are specified in templates. There are four files required in deploying a Niagara application, explained as follows.

1) Slots. In Niagara, a component is defined as a collection of slots, which specify the *properties*, *actions* and *topics* of events that the component is listening to. Although Niagara is essentially a Java runtime environment, the vendor, Tridium, made a customized process based on slot definitions. Slots are included in a comment section at the beginning of a class in java source. Thus the Java source file has to be preprocessed by a tool called *Slot-o-matic*, which translates the slot definition into actual java code that invokes BAJA (Building Automation Java Architecture) API. This has to be reflected in the artifact definition.
2) module-include.xml. This file indicates Niagara environment to register the components to an internal registry.
3) module.palette. This file specifies where to display the components in Niagara Development Environment, which puts the components in a tree-like structure.

The actual contents of these artifacts are out of the scope of this paper. Due to the similarity of simple file artifact specifications, we only exemplify the first two artifacts in Listing 9. Note that the language and compiler property of the slot artifact are respectively *slot* and *Slot-o-matic*. At deployment stage, this will indicate TOSCA environment to invoke the Slot-o-matic tool for the source code.

2) Sedona artifacts: Sedona framework is an open source IoT application environment. Compared to Niagara framework, Sedona is designed to keep the framework and application footprints small so that they can be deployed on resource-constraint devices. Applications are portable among devices with Sedona framework thanks to the Sedona Virtual Machine (SVM), which is similar to the concept of JVM. In fact, Sedona language is also similar to Java.

The artifacts required by the Sedona framework to deploy an application are more complicated. Figure 3 illustrates the artifact structure using a screenshot from the development environment. The function and type of each artifact is explained as follows.

1) Sedona source files. These files are indicated by the

Listing 9: Examples of Niagara artifact templates

```
<ArtifactTemplate id="uid:iot-niagara-file-1"
  type="SourceArtifact">
  <Properties>
    <FileArtifactProperties>
      <FileType>java</FileType>
    </FileArtifactProperties>
    <SourceArtifactProperties>
      <Language>slot</Language>
      <Compiler>Slot-o-matic</Compiler>
    </SourceArtifactProperties>
  </Properties>
  <ArtifactReferences>
    <ArtifactReference reference="src">
      <Include pattern="*.java" />
  ...
</ArtifactTemplate>

<ArtifactTemplate id="uid:iot-niagara-file-2"
  type="FileArtifact">
  <Properties>
    <FileArtifactProperties>
      <FileType>xml</FileType>
    </FileArtifactProperties>
  </Properties>
  <ArtifactReferences>
    <ArtifactReference reference="/">
      <Include pattern="module-include.xml" />
  ...
</ArtifactTemplate>
```

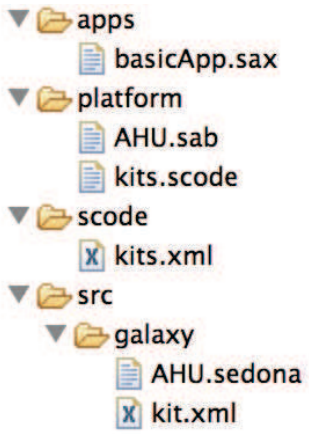

Fig. 3: Sedona artifacts

".sedona" extension and compiled by the *sedonac* tool.

2) kit.xml. Each Sedona component is called a kit. The kit.xml file defines the metadata for compiling sources into a kit.
3) kits.xml specifies the kits that are needed to build a deployable image, or the archive that can be deployed to SVM for running an application. The oBIX driver required to run the AHU application is compiled into the image.
4) *.scode. The image file built according to the specifications in kits.xml.
5) *.sax. Sax file constitutes the actual application configuration including for example the communication port and access credential.
6) *.sab. It is the executable binary file that is compiled according to the specification in the corresponding .sax file. The control logic of the AHU application is realized in this file.

All the required artifacts described above have to be correctly presented in the directory structure defined by the Sedona framework. Listing 10 presents examples of using TOSCA to specify the source, binary and deployable image to ensure that the files are correctly deployed.

Listing 10: Examples of Sedona artifact templates

```
<ArtifactTemplate id="uid:iot-sedona-file-1"
  type="SourceArtifact">
  <Properties>
    <FileArtifactProperties>
      <FileType>sedona</FileType>
    </FileArtifactProperties>
    <SourceArtifactProperties>
      <Language>sedona</Language>
      <Compiler>sedonac</Compiler>
    </SourceArtifactProperties>
  </Properties>
  <ArtifactReferences>
    <ArtifactReference reference="src">
      <Include pattern="*.sedona" />
  ...
</ArtifactTemplate>

<ArtifactTemplate id="uid:iot-sedona-file-6"
  type="BinaryArtifact">
  <Properties>
    <FileArtifactProperties>
      <FileType>sab</FileType>
    </FileArtifactProperties>
    <BinaryArtifactProperties>
      <Environment>SVM</Environment>
    </SourceArtifactProperties>
  </Properties>
  <ArtifactReferences>
    <ArtifactReference reference="platform">
      <Include pattern="*.sab" />
  ...
</ArtifactTemplate>

<ArtifactTemplate id="uid:iot-sedona-file-7"
  type="ArchiveArtifact">
  <Properties>
    <FileArtifactProperties>
      <FileType>scode</FileType>
    </FileArtifactProperties>
  </Properties>
  <ArtifactReferences>
    <ArtifactReference reference="platform">
      <Include pattern="*.scode" />
  ...
</ArtifactTemplate>
```

V. Discussion

TOSCA, as a newly established standard to counter growing complexity and isolation in cloud application environments, is gaining momentum in industrial adoption as well as academic interests. Following the first edition of TOSCA standard, we have showed that it is capable of specifying the basic constructs of IoT applications. By archiving the previous specifications and corresponding artifacts into a csar file, and deploying it in a TOSCA environment, the deployment of the

AHU application onto various gateways can be automated. This section will further discuss our experience gained from attempting to employ TOSCA in the IoT domain that is abundant of proprietary and largely heterogeneous frameworks.

1) **Acknowledge the heterogeneity**. In our previous work, we have proposed and prototyped the IoT PaaS framework that aims at more efficient and scalable IoT service delivery. The basic assumption is that IoT infrastructure is heterogeneous and will continue to be so. Thus, rather than proposing another "universal" architecture, we try to develop a methodology to easily integrate different domain-specific protocols and data models. That is *domain mediator* in the IoT PaaS architecture. Even worse than the situation in data exchange protocols, the deployment processes of an application can vary among IoT solutions even if the applications are realizing the same service. Following the same principle of avoiding proposing another "universal" management process, we leverage TOSCA to manage such heterogeneity in a coherent way—using a common vocabulary and syntax to describe application configurations and their deployment processes. As demonstrated in our AHU example, the node and relationship models can be shared for the same application, and the artifact models can be reused for gateways using the same software framework.
2) **Other TOSCA features**. This paper used several main features of TOSCA, namely *Node types*, *Relationship types*, *Artifact types*, *Properties*, *Interfaces* and corresponding templates. This assumes that TOSCA will process this service template in a declarative manner—the process of deployment is implicitly inferred according to the relationships expressed in the topology template. This will work for relatively simple applications as the simplified AHU. However, for more complicated applications, *Plans*, or the imperative approach, will be needed to explicitly invoke lifecycle operations and automate complex management processes. Furthermore, explicitly expressing the *Requirement* and *Capability* types will help the TOSCA environment to more accurately understand the dependencies between nodes, thus improve the reusability of models.
3) **Tooling and efforts of applying TOSCA**. As a new standard, the implementation of corresponding TOSCA tools is still in progress. The available tools have not realized all the standardized features. We are in the process of connecting the work-in-progress TOSCA environment with established IoT frameworks. We view this as a crucial effort in the early stage of the new standard. When the tool is matured and user contributions grow, more efforts will fall on collecting and improving models so that they can easily be reused. The tedious efforts of modeling each tiny aspect of IoT applications will eventually be rewarded with greater efficiency and reusability in the application management process.

VI. Related work

The research and application of TOSCA is still in its infancy. The early works are generally focused on exploring the possibilities of applying TOSCA for various management tasks, thus providing feedbacks to the standardization efforts and gaining experiences for industrial adoption. Wettinger et al. [13] presents several concepts that integrate both model-driven cloud management and configuration management. The goal of the overall approach is to combine the advantages of these service management paradigms based on TOSCA. Binz et al. [14] uses TOSCA to describe application topologies in a portable and manageable way. Based on this common TOSCA description the authors present an approach that merges two application topologies into one, to save resources by sharing similar components, but preserve the functionality of both applications. Breitenbucher et al. [15] proposes an approach that enables the management of composite applications and their deployment on a higher level of abstraction. Furthermore the authors show how high and low level management tasks can be implemented separately and fully automated applied to the respective applications, by facilitating the features of TOSCA. The work proposed in this paper is well in line with these early academic works on applying TOSCA to various scenarios. This paper presents the first effort of extending the application scope of TOSCA to an even more challenging area—IoT applications. It is also integral to our ongoing work on enabling the convergence of IoT and cloud through the IoT PaaS architecture.

This paper used Niagara and Sedona for demonstration. There are also other IoT frameworks that aim at facilitating device integration, protocol normalization and IoT application development. As these frameworks approach IoT infrastructures with different focuses, they are incompatible with each other and more of such frameworks are expected to emerge in the future.

IoTSyS[7][2] is a gateway concept that integrates various sensor and actuator systems, which can be found in current home and building automation systems. The integration middleware provides a stack of communication protocols for embedded devices based on various standards to support interoperability that gets directly deployed on 6LoWPAN devices. openHAB[8] presents an integration platform that operates on a higher level of abstraction. The architecture is based on an event bus in combination with a publish-subscribe pattern, realized on OSGi. To integrate any kind of device of an IoT infrastructure, abstract items are defined that represent these devices. In addition, bindings are used to bind items to concrete hardware, protocols or interfaces. This concept allows the platform to be vendor-neutral and hardware/protocol-agnostic. Since in IoT Systems most devices use their own proprietary communication stack and interfaces, it is challenging to offer the gathered data in a standardized way. Dawson-Haggerty et al. [16] proposes sMAP that tries to overcome this challenge by

[7]https://code.google.com/p/iotsys/
[8]http://code.google.com/p/openhab/

presenting physical information via RESTful interfaces using a simple JSON schema. This allows consumers to retrieve data, without the need to access the underlying infrastructure and dealing with proprietary formats. Based on sMap, Dawson-Haggerty et al. [17] presents BOSS, a distributed system that provides a collection of crucial, common and reusable services that enable the development of portable and robust applications for heterogeneous physical environment.

All introduced frameworks require considerable efforts to understand their application management process, and tedious manual configurations are a norm. Our work is complementary to the aforementioned frameworks. To be best of our knowledge, this paper presents the first attempt to address the IoT application deployment problem by using a domain-independent standard to explicitly specify the component topologies and management operations.

VII. Conclusion

In the face of the growing heterogeneity in IoT infrastructure and the need for more reusable and scalable IoT solutions, we propose to leverage cloud as a horizontal platform for managing the lifecycle of IoT applications. This paper presented the first efforts of using TOSCA, a new cloud standard, to formally describe the internal topology of application components and the deployment process of IoT applications. The feasibility of TOSCA for this purpose is demonstrated by describing the application components, relationships and artifacts of the AHU application using the first edition of the TOSCA specification. By inputing these descriptions to the TOSCA environment, the deployment process can be interpreted and automated.

As TOSCA is a young standard and the tools are still under development, we are in close contact with the TOSCA core team to accelerate the development process of core TOSCA tools and contribute the models and interfaces that are commonly needed in IoT applications. On modeling IoT applications, our future work is twofold. First is to produce more matured and detailed models for the applications we have already supported on IoT PaaS platform, especially those in building management domains. Second is to start to apply TOSCA to more IoT application domains along with the application development on IoT PaaS.

Acknowledgment

This work is sponsored by Pacific Controls Cloud Computing Lab (PC^{3}L)[9], a joint lab between Pacific Controls L.L.C., Dubai and the Distributed Systems Group of the Vienna University of Technology.

References

[1] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, Sep. 2012.

[2] M. Jung, J. Weidinger, W. Kastner, and A. Olivieri, "Building Automation and Smart Cities: An Integration Approach Based on a Service-Oriented Architecture," in *2013 27th International Conference on Advanced Information Networking and Applications Workshops*. IEEE, Mar. 2013, pp. 1361–1367.

[3] Q. Zhu, R. Wang, Q. Chen, Y. Liu, and W. Qin, "IOT Gateway: BridgingWireless Sensor Networks into Internet of Things," *2010 IEEEIFIP International Conference on Embedded and Ubiquitous Computing*, pp. 347–352, 2010.

[4] Tridium, "JACE Controller." [Online]. Available: http://www.tridium.com/cs/products_/_services/jace

[5] F. Li, M. Vögler, M. Claeß ens, and S. Dustdar, "Efficient and scalable IoT service delivery on Cloud," in *6th IEEE International Conference on Cloud Computing, (Cloud 2013), Industrial Track*, Santa Clara, CA, USA, 2013.

[6] OASIS, "Topology and Orchestration Specification for Cloud Applications (TOSCA)." [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

[7] T. Binz, G. Breiter, F. Leyman, and T. Spatzier, "Portable Cloud Services Using TOSCA," *IEEE Internet Computing*, vol. 16, no. 3, pp. 80–85, May 2012.

[8] F. Li, M. Vögler, S. Sehic, S. Qanbari, S. Nastic, H.-L. Truong, and S. Dustdar, "Web-Scale Service Delivery for Smart Cities," *Internet Computing, IEEE*, vol. 17, no. 4, pp. 78–83, 2013.

[9] OASIS, "Open Building Information Exchange (oBIX)." [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=obix

[10] ThereCorporation, "ThereGate." [Online]. Available: http://therecorporation.com/en/platform

[11] IETF, "Constrained Application Protocol (CoAP)." [Online]. Available: http://tools.ietf.org/html/draft-ietf-core-coap-08

[12] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services," *IEEE Transactions on Services Computing*, vol. 3, no. 3, pp. 223–235, Jul. 2010.

[13] J. Wettinger, M. Behrendt, T. Binz, U. Breitenbücher, G. Breiter, F. Leymann, S. Moser, I. Schwertle, and T. Spatzier, "Integrating Configuration Management with Model-Driven Cloud Management Based on TOSCA," in *Proceedings of the 3rd International Conference on Cloud Computing and Service Science, CLOSER 2013*. Aachen, Germany: SciTePress, 2013, pp. 437 – 446.

[14] T. Binz, U. Breitenbücher, O. Kopp, F. Leymann, and A. Weiß, "Improve Resource-Sharing through Functionality- Preserving Merge of Cloud Application Topologies," in *Proceedings of the 3rd International Conference on Cloud Computing and Service Science, CLOSER 2013*. Aachen, Germany: SciTePress, 2013.

[15] U. Breitenbücher, T. Binz, O. Kopp, and F. Leymann, "Pattern-based Runtime Management of Composite Cloud Applications," in *Proceedings of the 3rd International Conference on Cloud Computing and Service Science, CLOSER 2013*. Aachen, Germany: SciTePress, 2013.

[16] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. Culler, "sMAP: a simple measurement and actuation profile for physical information," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems - SenSys '10*. New York, New York, USA: ACM Press, Nov. 2010, p. 197.

[17] S. Dawson-Haggerty, A. Krioukov, J. Taneja, S. Karandikar, G. Fierro, N. Kitaev, and D. Culler, "BOSS: building operating system services," in *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, Apr. 2013, pp. 443–458.

[9] http://pc3l.infosys.tuwien.ac.at/