Journal of Computers Vol. 29 No. 6, 2018, pp. 264-273
doi:10.3966/199115992018122906026

IoT-ADL: An ADL for Describing Cloud IoT Applications

Miguel Zúñiga-Prieto[1], Daniel Rodríguez[1], Juan Rodríguez[1], Lizandro Solano[1], Emilio Insfran[2], Silvia Abrahão[2*]

[1] Department of Computer Science, Universidad de Cuenca, Av. 12 de Abril y Agustín Cueva, Cuenca, Ecuador
{miguel.zunigap, daniel.rodriguez, juan.rodriguezl, lizandro.solano}@ucuenca.edu.ec

[2] Department of Information Systems and Computation, Universitat Politècnica de València, Valencia, Spain
{einsfran, sabrahao}@disc.upv.es

Received 9 January 2018; Revised 21 January 2018; Accepted 22 January 2018

Abstract. IoT Applications coordinate the interaction of business services and IoT Devices, that automate business processes. The trend in building IoT Applications is to deploy their services on cloud platforms, which allow applications to take advantage of the cloud platform capabilities (e.g., large and dynamic storage and processing capacities). However, in spite of the growing number of IoT Applications, currently, their implementation is realized in an ad-hoc manner, without taking into account the heterogeneity of cloud platforms and IoT Devices. In this context, the interoperability among business services, IoT Devices and cloud resources available is a primary concern. In this article, we propose an Architecture Description Language (ADL) to specify the integration and interaction between IoT Devices and application services deployed in cloud environments, independently of their technologies. In addition, we propose an intelligent automation tool that provides: (i) a graphical editor for the ADL, and (ii) an automated code generator, that uses the architectural descriptions to generate software artifacts that implement the integration and interaction between the application services and IoT Devices. To illustrate the use of this ADL, the implementation of a Geographic Information System that supports the analysis of spatial data collected by air quality sensors, with geo-services deployed in the Google Cloud platform, is presented.

Keywords: architecture description language, cloud computing, GIS, IoT, model driven development, SoaML

1 Introduction

The trend in building information systems is to deploy their services on cloud platforms, using their large storage and processing capacities [1]. Cloud platforms enable developers to build applications that make a dynamic use of resources (e.g., hardware, software, networks, execution environment). In this context, applications use cloud resources according to their actual requirements, facilitating the payment based on consumption metrics (pay-per-use) [2]. On the other hand, applying IoT principles enables developers to build applications (IoT Applications) that interact with IoT Devices (things) in order to optimize or automate business processes and improve their accuracy. For example, processes involving data collection in Geographic Information Systems (GIS) are automated by using sensing devices managed through Internet [3].

Building IoT Applications deployed in a cloud computing platform not only presents challenges related to the technological heterogeneity of both cloud resources and IoT Devices [4], but also due to the lack of standards and development approaches. Heterogeneity of cloud environments generates a close dependence between service implementation and cloud provider technologies, limiting the portability between providers. Similar dependencies exist with IoT Device technologies.

There are proposals that integrate cloud computing and IoT paradigms in the development of GIS.

* Corresponding Author

Some suggest application architectures whose components are deployed in the cloud [1, 5-6]; while others use sensors under the IoT paradigm to perform data collection tasks [3, 7-9]. With regards to heterogeneity, proposals like [10-12] tackle it by proposing model driven development approaches; however, none of these proposals supports the design of the application architecture and the interaction between business services and IoT Devices. Developers need approaches that support the architectural design of IoT Applications and the implementation of interaction between business services and IoT Devices [3, 13].

The Model-Driven Development (MDD) approach proposes the use of models throughout the software development process, increasing the level of abstraction and allowing the generation of code through the application of successive model transformations. In this paper we propose an Architecture Description Language (ADL) for IoT Applications, which extends the ADL for Incremental Integration of Cloud Service Architectures (DIARy-ADL) [14]; providing it with the flexibility to describe specific characteristics of any domain, including the IoT domain. The ADL for Cloud IoT Applications support a MDD approach, where architectural models created with this editor are the input of model-to-text (M2T) transformations that generate software artifacts that implement the integration and orchestration between application services deployed in cloud environments with IoT Devices. This ease the integration of cloud application services with third party services, as well as with IoT Devices, enabling these devices to be connected to the Cloud. The applicability of this proposal is illustrated through the development of a GIS application, whose architecture consisted of geo-services deployed in the Google Cloud platform, and air quality sensors that support the data collection process under the IoT paradigm.

The remainder of the paper is structured as follows: Section 2 discusses existing proposals for IoT Applications development. Section 3 introduces the proposed ADL for IoT Applications. Section 4 presents an example to illustrate the feasibility of the ADL, and finally, Section 5 presents the conclusions and further work.

2 Related Work

Approaches to the interoperation with IoT Devices are proposed in [7] and [9], specifically, the implementation of Open Geospatial Consortium (OGC) standards-based data acquisition architecture, which defines mechanisms that allow the interaction of sensors and web services through interfaces and messages types. On the other hand, [3] and [15] propose an application architecture and protocols to support the integration of IoT Devices with services deployed in the cloud environments. However, these proposals do not provide mechanisms to support the architectural design of applications that interact with IoT Devices, nor mechanisms to handle the heterogeneity of IoT Devices during implementation and deployment. The application architectural design of IoT Applications is supported by [16], which proposes to use the architectural style Service Oriented Architecture (SOA) in order to deal with the interaction of application services and IoT Devices. However, does not support other development activities like implementation and deployment of application services on cloud environments.

The development of IoT applications is being supported by methods, programming languages and frameworks that address the heterogeneity of devices [17]. These approaches provide developers with a wide range of tools that allow the creation of IoT applications in an automated way. The MDD approach has been used in several studies to provide automation mechanisms on implementation code generation. For instance, [10-12] propose approaches and tools to raise the abstraction level of the IoT Application architecture, allowing the implementation of development automation mechanisms. Although these works deal with the heterogeneity of IoT Devices, they do not provide mechanisms that allow modelling the complete architecture of a system involving both IoT Devices and services deployed in cloud environments, lacking of solutions that support designing the interaction between them.

With regard to proposals that support architectural descriptions, we have identified ADLs for describing applications whose services will be deployed in cloud environments. Perez and Rumpe propose cloudADL [18] as the core element of a model based methodology for engineering cloud services. This methodology describes high-level cloud application architectures as interactive systems; however, the technological requirements specific for the cloud environment must be specified by using other architecture description languages, which poses new challenges to the architects that have to deal with the integration of different languages. StratusML [19] has been proposed as a modelling framework and domain specific modelling language for cloud applications. It provides multiple views and different

layers to address the cloud stakeholder's concerns, facilitates the visual modelling of adaptation rules and actions, and enables de generation of cloud artifacts (e.g., code, configuration). However, these proposals do not provide a solution for specifying characteristics of non-cloud architectural elements such as IoT Devices. Additionally, the learning and adoption of these languages requires from designers an additional effort. In order to facilitate both the cloud artifacts generation in later development activities and the interoperation between application services and IoT Devices, ADLs should allow architects to describe non-functional requirements at a high abstraction level as well as interaction protocol between services and IoT Devices.

3 ADL for IoT Applications in the Cloud

IoT Applications are generally composed of Web services [20], where principles of the architectural style SOA enable the integration services considering the heterogeneity of IoT technologies [16]. These principles promote the design and implementation of weakly coupled services; minimizing dependences among them by encapsulating particularities of implementation, requiring only a well-defined interface for communication between services. From a development point of view, service based applications are usually developed incrementally by building reusable services that may interoperate with each other.

The DIARy-ADL, an architecture description language for the incremental development and the dynamic architectural reconfiguration of cloud applications, allows developers to specify not only the architectural elements belonging to a software increment and the interaction protocol among services, but also the architectural impact of integrating these elements into the current application architecture. DIARy-ADL promotes the application of MDD and SOA principles and supports the specification of cloud domain non-functional requirements by using high abstraction level concepts, enabling the generation of implementation and deployment artifacts according to the architectural impact. However, it takes into account a fixed amount of requirements limited to services in the cloud domain.

In order to identify the type of services that conform an IoT Application deployed in cloud environments, we built a GIS whose services were deployed in the cloud and whose data collection process was supported by IoT Devices, then analyzed its design and implementation needs. We used the lessons learned to provide mechanisms that facilitate the architectural design. The analysis also involved the revision of similar works [3, 9, 15], identifying that IoT Applications are generally composed by three types of services:

- IoT Devices: Services offered by devices that within an application domain could automatically collect data, share information about their status, or run services with minimal human intervention. Increasingly, IoT Devices not only behave like simple sensors or actuators, but also provide restricted execution environments with limited processing, memory and storage capacities [21]. From the IoT perspective, each device is a potential provider of small services.
- Business Services: Services that: (i) provide specific business functionalities; (ii) offer general functionalities such as information integration, data processing, or visualization; or (iii) act as clients consuming other services. It includes software developed by third parties, deployed in a cloud environment, and integrated as part of the application architecture.

Cloud Resources: Technological resources offered as services by cloud providers (e.g., spatial database services provisioned in cloud environments, message queues). This kind of services may require the construction of another service, which acts as an intermediary for their access and management.

3.1 Description Language Abstract Syntax

In this paper, we propose the IoT-ADL, whose abstract syntax extends the DIARy-ADL abstract syntax, including concepts that allow the specification of IoT Applications' architectures. IoT-ADL provides architects with the flexibility to specify non-functional requirements of services in any domain, including the IoT domain (e.g., low energy consumption, e-mobility, scalability, availability, resilience). Additionally, it includes UML concepts related to sequence diagrams, allowing the description of complex interaction protocols among the services that make up the application architecture.

The IoT-ADL abstract syntax is specified by using the Ecore implementation of the EMOF [22] standard provided by the Eclipse Modelling Framework (EMF). Fig. 1 shows and extract of the IoT-ADL abstract syntax, where concepts inherit from the DIARy-ADL are depicted without background color;

whereas concepts proposed in this work are depicted with background color. The DIARy-ADL is based in SoaML [23], an OMG specification specifically designed for modelling SOA, facilitating service modelling and design activities; therefore its concepts, explained next, are similar to those proposed by SoaML.

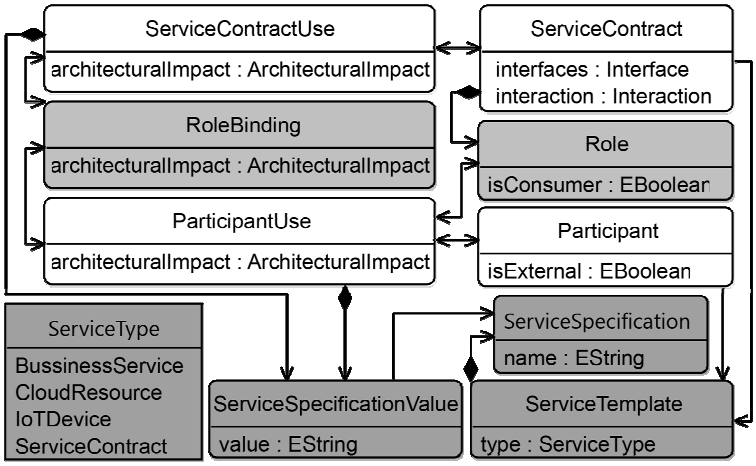

Fig. 1. Extract of the IoT-ADL metamodel

- *Services Architecture*: Describes how a set of Participants work together providing and consuming services for a particular business purpose or process. Its inner parts are Participants, Service Contracts, and Role Bindings. The IoT-ADL uses it to describe a IoT Application architecture that could be deployed as an inter-cloud software architecture.
- *Participant*: A participant may play a role of service provider, consumer, or both. According to the types of services identified in the previous section, the IoT-ADL allows architects to describe the following participants: *Business Services*, *Cloud Resources*, and *IoT Devices*. Architects specify the type of service through the attribute *type* corresponding to the related instance of the meta-class Service Template.
- *Service Contract*: Represents an agreement that describe how the communication (interaction) between Participants will be stablished. A Service Contract definition includes the following inner parts: (1) *Roles* that Participants involved in a service must fulfil in order to interoperate, (2) *Interfaces* that must be implemented by Participants in order to fulfil a Role. Interfaces describe *Operations* (*Messages* and their corresponding *Data Types*) provided and required to complete the service functionality. (3) *Interaction Protocol* that describes the interoperation between Participants without defining their internal processes. Service Contracts are implemented and deployed as orchestration services.
- *Role Binding*: Describes the Role that each Participant involved in a Service Contract plays.
- *Participant Use and Service Contract Use*: Participants and ServiceContracts may be reused, therefore a Participant Use references a Participant involved in a specific service, whereas a Service Contract Use explicitly specifies the use of the interoperation described in a Service Contract.
- The attribute *Architectural Impact* in Role Binding, Participant Use and Service Contract Use allows architects to specify the impact of integrating those architectural elements into the current IoT Application. Possible values are: *Reference* – used to tag elements already existing in the current application architecture that will interact with architectural elements of an increment, and will not change after integration –. *Add* – used to tag new elements to be integrated into de current architecture. *Modify* – used to tag elements already existing in the current Cloud Application Architecture whose implementation will change after integration –. Finally, Delete.

IoT-ADL extends DIARy-ADL abstract syntax and includes concepts to allow architects to specify non-functional requirements besides the cloud domain. IoT-ADL allows architects to define Service Templates (see Fig. 1) for each Participant and ServiceContract according to the attribute Service Type (i.e., Business Service, Cloud Resource, IoT Device, and Service Contract). Service Templates are composed by Service Specifications, which declare at a high abstraction level the service non-functional requirements. Architects assign values to non-functional requirements (Service Specification Value) for

each Participant Use and Service Contract Use according to their corresponding Participant and Service Contract.

3.2 Description Language Concrete Syntax and Semantics

Concrete syntax descriptions provide a notation that facilitates the construction, presentation, and maintenance of models in a language. Different notations can be provided (e.g., graphical, textual, tabular); in this context, a graphical editor (see Fig. 2) will reduce the perceived complexity of creating architectural models and manages their definition. We used Obeo Designer [24], an Eclipse plug-in to create the IoT-ADL editor (see Fig. 2). The editor was implemented following a multi tab design, where each tab is dedicated to describe an architectural element type: Service Templates (see Fig. 2(a)) IoT Application Architecture (see Fig. 2(b)), Service Contracts, Participants, Role Bindings, Interactions, Interfaces, and others. The editor includes a tool bar (or palette) that changes according the architectural element being specified (see Fig. 2(c)). In order to make interpretation of architectural models easier, the editor helps architects to identify the Service Type assigned to a Participant by using different graphical notations. For example, Fig. 2(b) shows the graphical notation used for Participants: Administrator whose type is Business Service, Sensor whose type is IoT Device, and Spatial Database whose type is Cloud Resource. Additionally, Service Contracts elements are depicted as ellipses whereas Role Bindings are depicted as lines that link Service Contracts with their involved Participants.

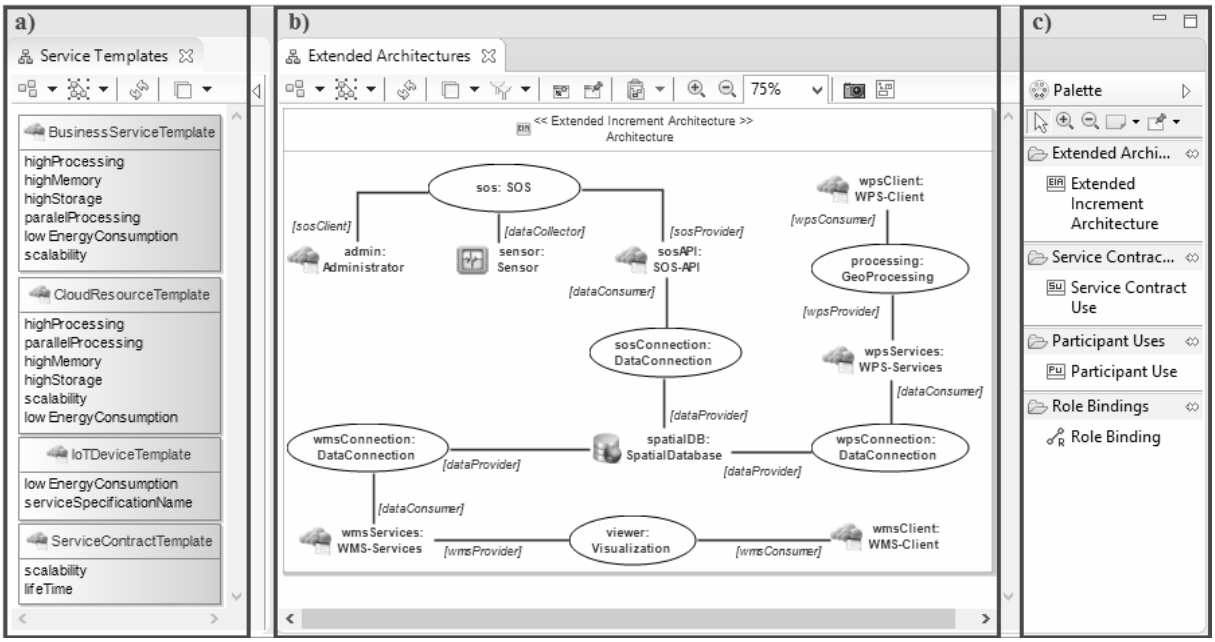

Fig. 2. IoT-ADL editor: IoT application architecture model

4 Using the IoT ADL

The IoT-ADL supports the activities described next.

4.1 IoT Application Architecture Specification

This activity is aimed at supporting architects in identifying the services that will conform the IoT Application as well as to describe interactions among them. Architects design the IoT Application Architecture Model (see Fig. 2(b)) using the IoT-ADL editor (see Fig. 2) creating instances of the IoT-ADL metamodel (see Fig. 1).

Architects begin by declaring Service Templates for Participants and Service Contracts. For example, Fig. 2(a) shows that, every time an architect assigns the template Business Service Template to a service (Participant) she/he will have to specify whether services are expected to require highMemory, highProcessing, parallelProcessing, or scalability support. Architects define at least a Service Template

per Service Type.

In the next step architects, identify the services that will conform the application and model them as Participants; assigning them a Service Template, which determine Participants' Service Type. Next, architects identify interaction needs among Participants, model them as Service Contracts, and assign them a Service Template. Architects specify Service Contracts by describing their inner parts: Roles, Interfaces, Interaction Protocol. Interface operations are described on the basis of standards; for instance, operations for sensor data collection could be described according to the Sensor Web Enablement (SWE) [7] standard. The Interaction Protocol among Participants that will play a Role is described by using a sequence diagram whose life lines are the Interfaces corresponding to each Role.

Once architects have described Participants and Service Contracts, they use them to describe the IoT Application Architecture (see Fig. 2(b)). Therefore, they create Participant Use and Service Contract Use elements and link them by using Role Bindings, specifying the Role that each Participant Use will play in the Service Contract Use in which it is involved. As IoT-ADL inherits DIARy-ADL, refer it for a deeper understanding of modelling software architectures.

Finally, architects analyze the nature of the work that each Participant/ServiceContract in the IoT Application Architecture model performs, and specify the non-functional requirements defined in the Service Template assigned to each Participant and Service Contract. Architects complete this step by giving values to the non-functional requirements declared in the Service Template's Service Specification. The specified values will be used for developers to take implementation and deployment decisions in further development phases.

4.2 Implementation Code Generation

This activity is aimed at supporting developers in generating the implementation code that orchestrate or coordinate the interaction among the IoT Application's Participants, as well as the implementation code that allows each Participant to interact. In this activity, developers create and execute M2T transformations that take as input the IoT Application Architecture Model and, according to the Architectural Impact specified, intelligently generate, modify, or delete the implementation code. This work extends the DIARy software infrastructure by providing M2T transformations plugins. Developers create these plugins the first time a new target technology is required, then reuse them. The steps to execute in this activity are.

- Generate Interaction Protocol: In this step, developers execute M2T transformations that generate, modify, or delete artifacts that implement the orchestration among Participants involved in a Service Contract. Transformation rules take as input the inner parts of Service Contracts generating an orchestration implementation from descriptions of Interaction Protocol (sequence diagram) and Interfaces. The generated artifacts will be deployed as an orchestration service, allowing the interaction and integration of the services involved in a Service Contract. Additionally, transformation rules generate a configuration file corresponding to orchestration services containing entries with information on the endpoints through which invoke operations of the Participants to be orchestrated.
- Generate Service Implementation Code: In this step, developers execute M2T transformations that generate, modify, or delete artifacts that implement the architectural element of type Interfaces corresponding to the Role each Participant plays. The generated artifacts implement the services offered for Participants and allow them to play a part of interactions.

4.3. Provisioning and Deployment

In this activity, developers take decisions related to execution environments to be provisioned in order satisfy the non-functional requirements specified for services offered by Participants and Service Contracts. Non-functional requirements are specified at a high abstraction level in instances of Service Specification Value concepts in the Application Architecture Model. Execution environments are provisioned either in the cloud or in IoT Devices depending on the Service Type. In the case of IoT it is necessary to assembly the electronic components and to install the software related to IoT Devices. In the case of cloud platforms, the provisioning of virtual cloud resources is performed.

Once execution environments have been provisioned, developers deploy services offered by Participants and Service Contracts in the corresponding cloud environment. Each service must be

accessible through access points in the form of a URL. It is important to make the necessary configurations to guaranty that all the URLs can be accessed via Internet, especially in the case of IoT Devices. To accomplish this, mechanisms such as public IP, domains, or VPNs can be implemented to connect cloud services to IoT services.

5 Illustrative Example

In this Section, an excerpt of an example is presented, that illustrates the use of the IoT-ADL. An environmental control company needs to implement an information system for analyzing air quality spatial data. For this, it decides to automate the process of collecting data of CO2 levels by using geo-positioned sensors, requiring manage data over Internet. The services' nature requires high storage and processing capabilities. To achieve this, the company has proposed the application architecture shown in Fig. 2(b), where the application services will be deployed using infrastructures provided by cloud providers.

5.1 IoT Application Architecture Specification

To design the application architecture, the previously mentioned editor was used. First, the Participants or services required are identified. Services of type: (i) Business Services, such as SOS services to manage observations collected from sensors (sosServices), WPS services for spatial processing (wpsServices) and WMS Services for map visualization (wmsServices); (ii) Cloud Resource services, such as Spatial Database Services (spatialDB); and (iii) IoTDevices for the collection of information by sensors (iotDevice). Next, the Service Contracts are defined.

Fig. 3 shows the interaction protocol description corresponding to the Service Contract SOS, whose messages where defined according to the SOS standard that is part of the OGS main standard SWE.

Fig. 3. Interaction protocol

5.2 Implementation Code Generation

After designing the application architecture, developers create M2T transformations to generate the implementation code of Participants and ServiceContracts. Acceleo, an Eclipse plug in, was used to create M2T transformations. Software artifacts related to Participant architectural elements were generated according to the Participant's Service Type, which are compatible either with a cloud provider technology or with a IoT Device technology. Software artifacts (e.g., Interfaces, Interaction Protocol) related to Service Contract architectural elements were generated as workflows according to a cloud provider technology. M2T transformations generated software artifacts as follows: Python scripts

compatible with Raspberry Pi devices, to be executed in the cgi-proxy mode of Apache, for IoT Devices; PHP files for BusinessServices; Data access source code, which enables access to database resources in Google Cloud Platform, for CloudResources; and Workflows (orchestration code) in PHP for ServiceContracts.

5.3 Running M2T Transformations

Google Cloud in IaaS mode was provisioned and used for hosting services (BusinessServices and CloudResource participants). The CloudSql Google Cloud platform service and its corresponding data access service was deployed.

Raspberry Pi 2 Model B device was selected for IoTDevice service type. The Raspbian operating system was installed with an apache server configured with a cgi-proxy running Python code. This IoT Device retrieves information from the MQ-135 sensor, using the ADC Converter MCP-3002 for analog reading. Service Contracts must be able to access the IoT Devices, therefore a VPN has been implemented on the virtual machine instance of the service contract, providing the ability to access the IoT Device through an IP. For the BusinessService participants with the parameter isExternal = false and the ServiceContract has been deployed in the cloud platform services in IaaS mode. The hardware and software requirements have been defined based on the tasks that the service has to fulfill.

6 Conclusions

Building IoT Applications to be deployed in the cloud is becoming a common practice; however, their development is realized in an ad-hoc manner, without taking into consideration the heterogeneity of cloud environments and IoT Devices, limiting the interoperability and portability between cloud and IoT platforms.

The proposed IoT-ADL, fills the gap in current development practices of IoT Applications deployed in cloud environments by providing a high-level and technology independent view of application architectures that describes the way application services and IoT Devices, that automate their processes, are integrated and interact. Additionally, IoT-ADL provides software architects with the flexibility to specify non-functional requirements of services in any domain, including the IoT and cloud domains. These specifications are used by developers to make decisions about the implementation and deployment in further development phases.

The feasibility of this language was demonstrated by designing and implementing a Geographic Information System, with geo-services deployed in the Google Cloud platform, that provides support for analysis of spatial data collected by air quality sensors. The solution presented in this work can be used to support other cloud providers and IoT Devices by defining the corresponding model transformations.

The main benefit of the proposed ADL is to allow specifying high level IoT and cloud nonfunctional requirements that must be taken into account for implementation and deployment. Additionally, the automation tool presented provides two main benefits: (1) a graphical editor to specify IoT Application architectures composed of IoT Devices, cloud services and business services; allowing the specification of interaction among them; and (2) a generating mechanism that uses the architectural descriptions to generate software artifacts that implement the integration and interaction between the application services and IoT Devices.

As future work, we plan to empirically validate the proposed ADL. We also plan to extend this proposal for automating the deployment on IoT Devices and generating code for other cloud providers.

Acknowledgments

This research is supported by the DIUC_XIV_2016_038 project, and the Value@Cloud project (MINECO TIN2013-46300-R).

References

[1] M.A. Bhat, B. Ahmad, Cloud computing: a solution to geographical information systems (GIS), International Journal on

Computer Science and Engineering 3(2)(2011) 594-600.

[2] Z. Liu, Typical characteristics of cloud GIS and several key issues of cloud spatial decision support system, in: Proc. the IEEE International Conference on Software Engineering and Service Sciences, ICSESS, 2013.

[3] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of Things (IoT): a vision, architectural elements, and future directions, Futur. Gener. Comput. Syst. 29(7)(2013) 1645-1660.

[4] A. Botta, W. De Donato, V. Persico, A. Pescapé, Integration of cloud computing and Internet of Things: a survey, Future Generation Computer Systems 56(2016) 684-700.

[5] K. Evangelidis, K. Ntouros, S. Makridis, C. Papatheodorou, Geospatial services in the cloud, Comput. Geosci. 63(2014) 116-122.

[6] P. Yue, H. Zhou, J. Gong, Geoprocessing in cloud computing platforms - a comparative analysis, International Journal of Digital Earth 6(4)(2013) 404-425.

[7] A. Bröring, J. Echterhoff, S. Jirka, I. Simonis, T. Everding, C. Stasch, S. Liang, R. Lemmens, New generation sensor web enablement, Sensors 11(3)(2011) 2652-2699.

[8] L.M. Guillén, J.M. Cartuche, J.G. Aungüisaca, J. Veintimilla-Reyes, Arquitectura para una red de sensores web basada en SWE (Sensor Web Enablement): Caso de estudio para la implementación en sensores hidrometeorológicos, Maskana Actas Cong 1(2016) 129-138.

[9] G. Sagl, M. Lippautz, B. Resch, Near real-time geo-analyses for emergency support: an radiation safety exercise, in: Proc. 14th AGILE International Conference on Geographic Information Science, 2011.

[10] M. Kabáč, A design-driven methodology for the development of large-scale orchestrating applications, [dissertation] Bordeaux, France: University of Bordeaux, 2016.

[11] X.T. Nguyen, H.T. Tran, H. Baraki, K. Geihs, FRASAD: a framework for model-driven IoT application development, in: Proc. IEEE World Forum Internet Things, WF-IoT 2015 - Proc., 2016.

[12] P. Patel, D. Cassou, Enabling high-level application development for the internet of things, J. Syst. Softw. 103(C)(2015) 62-84.

[13] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Communications of ACM 53(4)(2010) 50-58.

[14] M. Zúñiga-Prieto, E. Insfran, S. Abrahão, Architecture description language for incremental integration of cloud services architectures, in: Proc. 2016 IEEE 10th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments, MESOCA 2016, 2016.

[15] T. Ara, P. Gajkumar Shah, M. Prabhakar, Internet of things architecture and applications: a survey, Indian J. Sci. Technol. 9(45)(2016). doi: 10.17485/ijst/2016/v9i45/106507

[16] I. R. Chen, J. Guo, F. Bao, Trust management for SOA-based IoT and its application to service composition, IEEE Trans. Serv. Comput. 9(3)(2016) 482-495.

[17] R. Wenger, X. Zhu, J. Krishnamurthy, M. Maheswaran, A programming language and system for heterogeneous cloud of things, in: Proc. 2016 IEEE 2nd Int. Conf. Collab. Internet Comput, 2016.

[18] A.N. Perez, B. Rumpe, Modeling cloud architectures as interactive systems, in: Proc. the 2nd International Workshop on Model-Driven Engineering for High Performance and CLoud computing (MDHPCL 2013) co-located with 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013), 2013.

[19] M. Hamdaqa, L. Tahvildari, Stratus ML: a layered cloud modeling framework, in: Proc. IEEE International Conference on

Cloud Engineering (IC2E), 2015.

[20] E. Cavalcante, J. Pereira, M.P. Alves, P. Maia, R. Moura, T. Batista, F.C. Delicato, P.F. Pires, On the interplay of Internet of things and cloud computing: a systematic mapping study, Comput. Commun 89-90(2016) 17-33

[21] M. Vogler, J.M. Schleicher, C. Inzinger, S. Dustdar, DIANE-dynamic IoT application deployment, in: Proc. 2015 IEEE 3rd International Conference on Mobile Services, 2015.

[22] Object Management Group, OMG Meta Object Facility (MOF) core specification, version 2.4.1, 2013.

[23] Object Management Group, Service oriented architecture modeling language (SoaML) specification, version 1.0.1, 2012.

[24] E. Juliot, J. B. Benois, Viewpoints creation using Obeo designer or how to build Eclipse DSM without being an expert developer? <http://spotidoc.com/doc/197222>, 2010.