Supporting IoT Application Middleware on Edge and Cloud Infrastructures

Sven Akkermans, Stefanos Peros, Nicolas Small, Wouter Joosen, Danny Hughes

imec-DistriNet, KU Leuven Celestijnenlaan 200A, 3001, Leuven, Belgium {first}.{last}@cs.kuleuven.be

Abstract. The Internet-of-Things (IoT) is evolving from classic networks to cloud-style heterogeneous infrastructures, including both edge and cloud entities. As a result, application creators, system managers and infrastructure providers are now distinct stakeholders. The problem is that current IoT middleware solutions are: (1) insufficiently expressive. limiting the specification of application aspects such as Quality-of-Service, and/or (2) require infrastructure-specific knowledge from application creators to meet their application requirements. This paper argues that it is essential for middleware to enable application creators to define functional and non-functional aspects of applications since middleware require this information to correctly deploy and manage applications. Accordingly, expressive abstractions are needed to specify applications as service compositions while hiding the increased complexity of emerging IoT infrastructures. Concretely, the contributions are: (1) a classification of functional and non-functional aspects of IoT applications and (2) requirements for IoT application middleware to simplify application management for different stakeholders.

Keywords: Internet-of-Things, Application Middleware, Cloud Computing

1 Introduction

The Internet-of-Things (IoT) is changing from Wireless Sensor Networks (WSNs) connected with remote cloud servers to cloud-style heterogeneous infrastructures through new paradigms such as edge computing and high-level specification languages. Smart devices are becoming part of a multi-tier IoT infrastructure with both edge (e.g., end-devices, gateways) and cloud entities capable of supporting applications composed out of services [6].

This evolution leads to three distinguishable categories of stakeholders: application creators, system managers and infrastructure providers. Application creators are application-domain experts with little middleware or infrastructure knowledge. System managers provide application creators with a middleware to specify, deploy and manage their applications. Infrastructure providers supply the physical and virtual resources of the IoT infrastructure.

N. Herzberg, C. Hochreiner, O. Kopp, J. Lenhard (Eds.): 10th ZEUS Workshop, ZEUS 2018, Dresden, Germany, 8-9 February 2018, published at http://ceur-ws.org/Vol-2072

This distinction between stakeholders challenges middleware to provide expressive abstractions for specifying functional and non-functional application aspects while hiding infrastructure complexity. To the best of our knowledge, current middleware solutions provide limited support for application runtime specifications or rely on the infrastructure knowledge of the application creator to make deployment decisions [2].

This paper decomposes IoT applications into a generic classification of functional and non-functional aspects as a first step to tackle this challenge. From this classification, we propose requirements for middleware to support expressive and transparent creation, deployment and management of applications on the edge and cloud IoT infrastructure. Modelling IoT applications as compositions of functional and non-functional aspects allows application creators to specify the function and runtime characteristics of their applications and makes it simpler for middleware to match applications to the available resources.

In summary, the contributions of this paper are (1) a classification of generic functional and non-functional aspects of IoT applications, and (2) requirements for middleware to support expressive and transparent creation, deployment and management of IoT applications on edge and cloud infrastructures.

2 Classification of IoT Application Aspects

This section presents a high-level classification of IoT application aspects by extracting application specifications and requirements from related research. We analysed 26 papers from previous EWSN, IPSN and SenSys conferences and 36 papers focused on keywords around WSNs, IoT applications, Edge and Cloud computing¹. We classify high-level application aspects into two categories, similarly to [1]: functional elements and non-functional properties. Functional elements express pure functionality, e.g., what the functions do. Non-functional properties express runtime properties of the application, e.g., how the functions are performed. The values of their configuration parameters depend on the capabilities of the available resources in the infrastructures and middleware in question. The goal is to provide a representative set of values to guide middleware in using the classification of functional and non-functional application aspects.

Functional Application Elements. Representative functional elements can be obtained from the cross-cut of functional concepts listed in the surveyed papers. Due to space concerns, we limit discussion here to well-cited WSN application papers [4,7,11] and IoT application frameworks [12,14].

Bai et al. [4] focus on sensor networks to develop a taxonomy of WSN archetypes. They extract eight key application concepts: *mobility, initiation of sampling, initiation of data transmission, actuation, interactivity, data interpretation, data aggregation and homogeneity.* Rahman et al. [14] survey the literature for IoT frameworks and map functionality to physical devices. They define twelve

¹ Available online at: goo.gl/Mw9p1p [Last Access: January 2017]

42 Sven Akkermans et al.

typical IoT functions: sense, actuate, profile, device management, control, application, API, discovery, storage, vertical analytics, horizontal analytics and translation. **Oppermann et al.** [11] survey a decade of WSN applications and present a taxonomy of WSNs. They focus on WSN design aspects for common applications: goal, sampling approach, sensed phenomenon, data rate, heterogeneity, mobility, connectivity, processing, storage, services and communication primitives. **Patel et al.** [12] present a development methodology that separates IoT application development, separated into four concerns (i.e., domain, functional, deployment, and platform). They list five concepts that map to software components which encapsulate system functionality: sensor, storage, computation, actuator and user interface. **Greenstein et al.** [7] propose a sensor network application construction kit that is based on smart application service libraries. Their library contains six components, which together create applications: sense, aggregate, transmit, route, process data and storage.

The analysis of these papers results in six functional elements and associated configuration parameters that are commonly present in IoT application design. Table 1 provides an overview of the functions with sample configuration values. **Data Source** is the function that creates data, e.g., through measurements. Location determines where the function has to be done. Data type dictates the type of the data. Initiation method determines when the function performs. Result transmission governs when the function result is transmitted. Actuator is the function that performs physical actions. Action type is the type of performed physical action. **Store** is the function that stores data. Data gathering specifies how the function receives data. Storage processing determines how data is processed before being stored. Storage method governs how the data is stored. **Processor** is the function that processes data. Processing method determines what is done with the data. **Interface** is the function that presents data to the front-end. Data view determines how the data is presented. Function Manager is a meta-function that manages other functions. Query functions retrieves metadata from functions. Control functions controls other functions.

Function	Configuration Parameter	Configuration Parameter
Data Source	Location (Spot, Local, Mobile,)	Data Type (Motion, Weather,)
	Initiation Method (Event-Driven, Periodic,)	Result Transmission (Passive, Active,)
Actuator	Location (Spot, Local, Mobile,)	Action Type (On/Off, Move, Buzz,)
	Initiation Method (Event-Driven, Periodic,)	
Store	Data Gathering (Passive, Active,)	Storage Processing (Filter, Compress,)
	Storage Method (Caching, Persistent,)	
Processor	Initiation Method (Event-Driven, Periodic)	Data Gathering (Receive, Request,)
	Processing Method (Sort, Trigger, Translate,)	Result Transmission (Passive, Active,)
Interface	Data Gathering (Receive, Request,)	Data View (Event-Driven, Periodic,)
Function Manage	r Query Functions (Discover, Expose,)	Control Functions (Configure, Migrate,)

Table 1. Functions and configuration parameters with possible values.

43

Non-Functional Properties. Middleware requires application creators to specify properties during application specification to deploy and manage their applications. The middleware determines the possible specifications of the properties and how to meet them based on its managed infrastructures.

We discuss some identified papers though research in non-functional aware deployment for Cloud/Edge IoT applications is sparse [2]. Horre et al. [9] identify quality aware deployment specifications as an important aspect of software deployment for multi-purpose WSNs, but only consider coverage. Heinzelman et al. [8] propose a middleware solution that supports QoS for applications on top of WSNs. However, they integrate the QoS application requirements with sensor network management, tightly coupling QoS support with the underlying infrastructure, limiting it to sensor networks and neglecting edge and cloud infrastructures. Brogi et al. [5] propose a model to support QoS-aware deployment of IoT applications over Fog infrastructures, but consider only bandwidth and latency as metrics. These papers partially list non-functional metrics to evaluate their models, but lack a classification of non-functional properties for the IoT.

Table 2 lists the extracted non-functional properties, how they can be specified and the main factors of the infrastructure that influence them. The infrastructure providers determine how their infrastructure influences the non-functional property at hand.

Property	Possible Specification	Infrastructure Factors
Latency	Upper bound in time units	Network
Lifetime	Range in time units or run count	Resource Usage
Dependability	Percentage of uptime	Verification
	Degree of redundancy	Redundancy
Security	Required encryption level	Encryption
	Data placement and movement	Data Locality
Cost	Monetary unit per resource usage or time unit Pricing Model	

Table 2. Non-functional elements, their specifications and main infrastructure factors.

Latency is the communication delay between the functions of an application, specified in time units and commonly upper bounded. The main influencing factor is the infrastructure network where the communicating functions reside. The network protocol and topology direct latency in several ways. Influencing factors are routing strategies, hop depth, crossing network boundaries and co-locating entities to avoid communication overhead [10,13]. Lifetime of a function refers to the duration (in time units) or the amount of times it performs work [13]. Infrastructure resource usage is the main influencing factor for lifetime. Edge and cloud infrastructures can have battery-powered resources, such as IoT end-devices, or resources with limited usage allowance, such as leasing time on cloud VMs. Dependability is the property of a function to perform correctly. It can be expressed as desired uptime or through the degree of redundancy. The infrastructure increases dependability by way of verification, to ensure functions perform as

44 Sven Akkermans et al.

intended, and redundancy, to protect against failure. Function working verification can be done through heartbeats for uptime and message acknowledgements to ensure packet delivery and redundancy through multiplexing a function, for example, using multiple sensors for more sources. Security, as well as privacy, of IoT applications is a broad subject, mostly concerned with the data of the application. Data is more secure and private when encrypted or handled in trusted domains. Therefore, we abstract data placement, movement and its encryption as specifications for privacy and security. Middleware influences this through function placement in the infrastructure. Placing functions on encryption-featured resources and bounding data or data flows to or between trusted resources or areas, avoid unsecured resources having access to the data. Cost refers to the cost of the application to run on the infrastructure. Two methods that can be used to express cost requirements of an IoT application, based on research around cloud pricing models [3] are: pay-per-use and subscription-based. In the former, the application is charged based on resource usage while in the latter it is charged on a time basis. Infrastructure determines the pricing model and monetary unit.

3 Requirements for IoT Application Middleware

This section defines requirements and guidelines for IoT application middleware based on the classification proposed in Section 2 and the distinction between stakeholders. System managers need to provide expressive application specifications to application creators which shield them from the underlying infrastructures of the infrastructure providers. This is simplified through blueprints, proposed abstractions of application specifications, essentially a contract from the application creator to the middleware. System managers use blueprints to inform application creation, deployment and management.

Requirements. In this section, we identify four high-level requirements for IoT application middleware. These requirements are defined with respect to the needs of the application creator and the required support of the underlying middleware:

- R1: Application creators should be able to specify *what* an application does [1]. Middleware should provide the functions that are possible on the infrastructures to compose applications.
- R2: Application creators should be able to specify how an application operates [1]. Middleware should provide the ability to specify the non-functional properties it can support.
- R3: Application creators have their applications deployed and managed transparently to the underlying infrastructures. Middleware should autonomously insert additional functions if necessary in the infrastructure to support function interoperability and non-functional properties.
- R4: Application creators should only be able to specify functions and nonfunctional properties that are possible in the provided infrastructures. Middleware should maintain a consistent view of the capabilities of its resources and infrastructures and relate them to functions and non-functional properties.

45

3.1 Blueprints

In this section, we propose *blueprints*, an application specification abstraction based on the classification in Section 2 and the requirements in Section 3. Three distinct elements can be derived from **R1**, **R2** and **R3** for application specification: user functions, middleware functions and non-functional properties. Application creators compose blueprints through a combination of user functions, essentially services, and non-functional properties. The middleware, with a view of the resources and infrastructures, as specified in **R4**, provides suitable options to application creators when creating blueprints. For instance, the middleware should enable creating a Data Source function with a mobile location option on a mobile tracking resource in the infrastructure. User functions are specified by the application creator while middleware functions are autonomously inserted by the middleware. Middleware presents the possible functions, configuration parameters and values to the application creators, which they specify and connect. Middleware functions can be necessary to support non-functional properties or other functions, depending on the infrastructure, and so should be autonomously inserted by the middleware. For instance, communicating functions on different resources may need an intermediate Processor function for translation or functions might need to be periodically exposed by a Function Manager to ensure reliability.

Blueprints support stakeholders in three phases of the application life-cycle. For creation, blueprints remove the burden of programming from application creators by providing generic functions and allow expressing non-functional properties. This is motivated by basing functions and properties on a wide range of existing research. For deployment, blueprints provide the middleware with the application requirements it needs to intelligently deploy the application on the suitable resources. For management, blueprints allow to balance changing infrastructural concerns with ongoing application requirements. For instance, in case of unexpected resource failures, replacement functions can be inserted.

4 Discussion and Conclusion

This work supports middleware in the management of IoT applications across multi-tier infrastructures. IoT middleware can implement blueprints by supporting the listed functions, essentially services, and non-functional properties on their managed infrastructure. This requires middleware to have suitable configurable programming constructs (e.g., Python scripts, Contiki ELF modules, ...) for functions and the ability to define non-functional properties on top of them. For instance, if the middleware supports application specification in an XML-language (e.g., as in [9]), the discussed functions and properties should be part of the language the application creators use. The resulting application specification is a blueprint and enables the middleware to deploy and manage the application. In practice, blueprints are currently being implemented on Niflheim, an end-to-end middleware for applications on a multi-tier IoT infrastructure [15].

This paper contributes a classification of functional elements and non-functional properties, essential for application specification, and a set of requirements for

46 Sven Akkermans et al.

middleware to support the management of IoT applications, transparently to the emerging edge and cloud infrastructures. This research is based on existing literature which covers a wide range of applications. To conclude, this work supports IoT application middleware in application specification through blueprints and by listing necessary requirements to foster application use on IoT infrastructures.

References

- Aldinucci, M., Danelutto, M., Kilpatrick, P.: Autonomic management of nonfunctional concerns in distributed parallel application programming. In: 2009 IEEE International Symposium on Parallel Distributed Processing. pp. 1–12
- Arcangeli, J.P., Boujbel, R., Leriche, S.: Automatic deployment of distributed software systems: Definitions and state of the art. Journal of Systems and Software 103, 198–218 (2015)
- Artan Mazrekaj, I.S., Sejdiu, B.: Pricing schemes in cloud computing: An overview. International Journal of Advanced Computer Science and Applications 7(2) (2016)
- Bai, L.S., Dick, R.P., Dinda, P.A.: Archetype-based design: Sensor network programming for application experts, not just programming experts. In: 2009 International Conference on Information Processing in Sensor Networks. pp. 85–96 (Apr 2009)
- 5. Brogi, A., Forti, S.: Qos-aware deployment of iot applications through the fog. IEEE Internet of Things Journal (2017)
- Chiang, M., Zhang, T.: Fog and IoT: An Overview of Research Opportunities. IEEE Internet of Things Journal 3(6), 854–864 (Dec 2016)
- Greenstein, B., Kohler, E., Estrin, D.: A Sensor Network Application Construction Kit (SNACK). In: Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems. pp. 69–80. SenSys '04, ACM
- 8. Heinzelman, W.B., Murphy, A.L., Carvalho, H.S., Perillo, M.A.: Middleware to support sensor network applications. IEEE network 18(1), 6–14 (2004)
- Horre, W., Michiels, S., Joosen, W., Hughes, D.: QARI: Quality Aware Software Deployment for Wireless Sensor Networks. pp. 642–647. IEEE (2010)
- Mahmud, R., Buyya, R.: Fog computing: A taxonomy, survey and future directions. CoRR abs/1611.05539 (2016)
- Oppermann, F.J., Boano, C.A., Romer, K.: A Decade of Wireless Sensing Applications: Survey and Taxonomy. In: Ammari, H.M. (ed.) The Art of Wireless Sensor Networks, pp. 11–50. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
- 12. Patel, P., Cassou, D.: Enabling high-level application development for the Internet of Things. Journal of Systems and Software 103(Supplement C), 62–84 (May 2015)
- Priyantha, N.B., Kansal, A., Goraczko, M., Zhao, F.: Tiny web services: Design and implementation of interoperable and evolvable sensor networks. In: Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems. pp. 253–266
- Rahman, L.F., Ozcelebi, T., Lukkien, J.J.: Choosing Your IoT Programming Framework: Architectural Aspects. In: 2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud). pp. 293–300 (Aug 2016)
- Small, N., Akkermans, S., Joosen, W., Hughes, D.: Niflheim: An end-to-end middleware for applications on a multi-tier iot infrastructure. In: 2017 IEEE 16th International Symposium on Network Computing and Applications (NCA). pp. 1–8