QoS-by-Design in reconfigurable IoT ecosystems

Michiel Willocx, Ilse Bohé, Vincent Naessens
MSEC, imec-DistriNet
KU Leuven, Technology Campus Ghent
Gebroeders Desmetstraat 1, 9000 Ghent, Belgium
firstname.lastname@kuleuven.be

Abstract—**Attractive Internet-of-Things (IoT) ecosystems need to cope with fast edge hardware evolutions. Offering flexibility to IoT infrastructure owners when disrupted or broken sensors and actuators need to be replaced is of major importance. It must be possible to plug in commercial-of-the-shelf alternatives that are probably more reliable and/or cheaper at a given time. Decoupling the application from the underlying infrastructure during software design increases flexibility and reconfigurability. Application developers no longer create applications tailored to one specific sensor or actuator. However, quality concerns imposed by the application may restrict feasible paths that can be taken by infrastructure providers. Hasty decisions can result in improperly functioning IoT ecosystems. This paper proposes a structured approach to embed Quality-of-Service (QoS) challenges in the development and operations cycle of advanced IoT environments, thereby incorporating clear separation-of-concerns between app developers and infrastructure providers. Its practicality is demonstrated by means of a smart office case.**

I. Introduction

The digital transformation is omnipresent in many sectors. In this decade, an emergent number of companies are exploring the opportunities of equipping industrial and logistic environments with Internet-of-Things (IoT) technologies, or extending existing work-related as well as leisure software applications with edge technologies. For instance, in the transport sector, smart IoT applications support GPS driven vehicle tracking-and-tracing, as well as cargo monitoring by sensing temperature, humidity and vibration values.

The advantages of connecting sensors and actuators to computer systems are well known. However, many system integrators are currently struggling to build sustainable IoT ecosystems cost-efficiently. During the first IoT wave, IoT applications were built and tailored to one specific sensor or actuator manufacturer. Sensor-specific code was often intertwined with the business logic in IoT apps. The latter negatively impacts the software maintainability, and results in unexpected and costly vendor lock-ins. Today, the software lifetime exceeds the lifetime of hardware devices in many IoT ecosystems. Front-end devices break due to harsh environmental conditions or become obsolete after a relatively short period. However, in many IoT systems, broken sensors can only be replaced by new ones of the same type, model and vendor although more qualitative or cheaper alternatives are already on the market at that time.

Embracing flexibility and reconfigurability as key concerns during the design process of IoT ecosystems is essential to develop sustainable IoT applications. Application-centric development aims at combatting the vendor lock-in trap. This design paradigm postulates that IoT technology selection – and in particular the edge devices – should no longer be done as the first step in the IoT design and development process. On the contrary, during the design of IoT apps, sensor and actuator technology should be abstracted. Infrastructure providers are responsible for selecting and installing the edge devices required by the applications at deployment time.

A layered software architecture can separate business logic from the underlying IoT infrastructure. Uniform technology-agnostic interfaces can be called from the business logic layer. The underlying layers can subsequently bind the calls to concrete IoT devices, given the appropriate plugins are loaded. Although these design tactics facilitate reconfigurability, the options are not unlimited. For instance, a broken sensor monitoring the temperature of boxes during organ transport cannot be replaced by whatever other temperature sensor on the market. Quality-of-Service (QoS) properties restrict the feasible options. For instance, hard constraints can be imposed on acceptable polling frequency and accuracy. Soft constraints can be imposed on battery life. Similarly, fall detection systems need to be reliable, and guarantee data transmission within a small time interval.

Contribution. This paper proposes an approach to tackle QoS challenges during the development and operations cycle in advanced Internet-of-Things environments. A layered architecture consisting of an IoT device virtualization layer that manages the set of devices in a particular IoT ecosystem facilitates reconfigurability. Each device is annotated with *required* and *desirable* QoS properties during the design phase and an *IoT device catalogue* keeps relevant QoS properties for existing IoT products. A software engine aids the infrastructure provider during the selection of concrete IoT technologies. The methodology is applied to a smart office case study. Note that this contribution focuses on tackling QoS concerns by integrating commercial-off-the-shelf (COTS) sensors and actuators, not by designing novel edge devices tailored to meet advanced reliability needs.

The remainder of this paper is structured as follows. Section 2 points to related work. Section 3 introduces the case study. Section 4 describes the IoT device virtualization approach. The approach is presented in Section 5 followed by a discussion in Section 6. This paper ends with conclusions.

II. Related work

QoS has been a subject to many research in IoT. Gubbi et al. [1] point to it as one of the major open challenges. White et. al. [2] show that most existing research focuses on improving sensor and actuator technologies based on a SLA of more than 150 papers.

Network layer contributions evaluate the overall performance of networks [3], [4], and present strategies to improve the QoS provided by one specific network standard or protocol [5]–[7]. Parameters of interest are network reliability, delays, throughput…

Other work assesses the QoS provided by specific sensors and actuators. Accuracy and reliability are important QoS parameters. For example, Christiansen et. al. [8] evaluate the accuracy of a glucose sensor by comparing the readings against reference values. Zhang et. al. [9] develop and test an ultrasensitive humidity sensor that demonstrates rapid response times. Wang et. al. [10] compare the accuracy of several commercial wrist-worn heart rate sensors.

Many *cloud solutions* [11], [12] were proposed over the last years. In this space, QoS research aims at increasing the reliability and decreasing delays of edge devices by revising the registration, orchestration and composition of sensors.

Last, QoS concerns are also tackled in *wireless sensors networks (WSNs)* and fog computing [13]–[15]. Important QoS concerns in these settings are related to networking (e.g. managing the delay and throughput between different nodes, automatic network configuration), resource management (e.g. computing power and battery life), availability and load balancing between individual nodes. Often, a specific application domain is tackled. Examples are industrial IoT [16], [17] and health care [18], [19].

In nearly all aforementioned work, QoS properties are improved by novel hardware or software features on the edge devices. On the contrary, our approach considers edge devices as a black box, and rather focuses the proper selection and integration of COTS products in IoT ecosystems. The choice can be constraint by both functional and non-functional concerns imposed by the application.

III. Case study

To demonstrate the practical feasibility, our approach is applied to lighting experience in smart office environments. However, the methodology can easily be applied to other application domains.

A proposed tablet application aims at controlling light bulbs in a meeting room, and logging information about their status. The offices are equipped with sensors that monitor room brightness. Once brightness falls behind a predefined threshold, the lamps need to be turned on. It must be possible to dim the lamps in meeting rooms when presentations are given. The employee needs to interact with the tablet application to steer the light intensity. Besides the functional behaviour, a set of quality requirements are defined. First, minimal brightness thresholds can be imposed for each meeting room or office. Second, it must be possible to dim the lights. Finally, the response delays must be acceptable. In case room brightness falls behind a certain threshold as well as in case of user input, the light bulb intensity should be modified within a certain time interval.

IV. Device Virtualization

Our work builds upon SMIoT [20], an architecture for developing maintainable IoT applications. The architecture proposes multiple abstraction layers to build versatile IoT applications, and its practicality is already demonstrated through the development of various IoT applications in health care, fleet management, access control… A high degree of flexibility and reconfigurability is achieved by the virtual IoT device layer. In this layer, a uniform interface is defined for each IoT device type that is relevant for the IoT ecosystem under design. The uniform interface defines operations that can be performed on that type of IoT device. For instance, uniform interfaces can be defined to monitor and/or control a fall detector and camera in an ambient assisted living ecosystem. Similarly, in a smart living environment, uniform interfaces can be defined to steer smart lamps and to control environmental parameters such as humidity, air quality… The uniform interfaces are technology-agnostic. Note that our demo app consists of two IoT device types, namely *Lamp* and *LightSensor*. To support a particular IoT technology (f.i. a 112FallDetection bracelet, an Axis IP camera, a Philips Hue light bulb…), a plugin must be developed. The plugin implements the uniform interface and handles communication between the app and the IoT device. For instance, a *Lamp* contains methods for turning the lamp on and off, and changing its color and brightness. For each supported technology, a model/vendor specific software plugin implements the uniform interface and handles the communication with the device.

The plugin is bound to the application at configuration time (or even at runtime). The approach facilitates the job of application developers. They are no longer confronted with the technical details of particular sensor or actuator technologies. Moreover, this approach is proven to be effective to avoid the vendor lock-in trap. The architectural approach can easily cope with new IoT hardware that is coming at a fast pace to the market. Supporting the most feasible hardware technology only requires that a new plugin is added to the application.

Although supporting cost-efficient integration of alternative IoT technologies is a step forward towards sustainable IoT applications, the alternatives are constraint. The freedom of choice can be limited by QoS requirements. For instance, in some applications, delays must be kept under control. In others, accuracy may not fall behind an unacceptable threshold. This work exactly copes with these QoS requirements, and complements the current SMIoT work.

V. Approach

The proposed approach consists of four steps, namely (a) eliciting relevant QoS) properties, (b) creating a device catalog, (c) building the application and (d) selecting/coupling feasible IoT devices.

A. Elicitation of relevant QoS properties

For each device type in the IoT ecosystem, a list of relevant QoS properties is compiled along with one or more expected value types and their semantics, together with a list of supported units. For instance, the lamp brightness can be expressed in *lumen* and an *integer* value can be assigned as maximum value. Other properties do not have a value type. They just reflect about the presence or absence of a property in a particular technology. For instance, the brightness and/or color of certain lamps can be changed. For others, this is impossible. The units allow for automated translations between different units in tooling support. Table I lists the relevant QoS properties of the two device types (i.e. *Lamp* and *LightSensor*) in our demo app. Note that the amount of quality properties that can be assigned to a device type can be huge. For instance, besides the quality properties currently kept in the table (i.e. brightness, color and response delay), others could be added. For instance, expected lifetime (in terms of lighting hours) and humidity resistance might be relevant properties in other settings. At least the quality properties that are crucial for the application under design should be included.

Device type: Lamp		
property	**value type**	**supported units**
change_brightness	-	-
change_color	-	-
brightness	maxValue(int)	lm
response_delay	maxValue(int)	msec, sec
Device type: LightSensor		
property	**value type**	**supported units**
response_delay	maxValue(int)	msec, sec
polling_frequency	minValue(int), maxValue(int)	/sec, /min, /hr, /day
precision	value(double)	lx
data_range	minValue(int), maxValue(int)	lx

TABLE I
RELEVANT QOS PARAMETERS FOR A LAMP AND A LIGHTSENSOR IN THE SMART OFFICE APP.

B. Device catalog creation

During device catalog creation, a set of products are assigned to each device type. For each product, the list of QoS properties is instantiated together with a pointer to a plugin that can be used to add the product to the app without any substantial development effort. Absence of such a plugin means that the app developer still has to bind the product to the application. Table II shows a device catalog for the smart office app. Analogue tables can be constructed for the other device types and products. They can be reused across multiple application domains. Note that some quality properties (like *brightness* of *Lamps*) can be extracted from the product specification. Others – namely the starred ones – are defined by experimental set-ups.

Lamps	Philips Hue White	Philips Hue White and Color	IKEA TRÅDFRI
change_brightness	✓	✓	✓
change_color	$\mathcal{X}$	✓	$\mathcal{X}$
brightness	800 lm	800 lm	960 lm
response_delay	1 sec*	1 sec*	1 sec*
plugin	✓	✓	$\mathcal{X}$
LightSensors	Arduino Bluetooth	Arduino LoRa	Versasense
response_delay	–	1 sec*	1 sec*
polling_frequency	1/sec - ...*	12/hr - ...	6/min - 1/day
precision	1 lx	1 lx	0,01 lx
data_range	0-2000 lx	0-2000 lx	0-16496 lx
plugin	✓	✓	✓

TABLE II
EXAMPLE OF PRODUCT DEFINITIONS FOR BRIGHTNESS SENSORS. BOTH ARDUINO SETUPS ARE EQUIPPED WITH A *Grove brightness sensor*[1].

C. Application design

As described before, the application developer relies on virtual IoT device calls to interact with sensors and the actuators. Those calls are technology-agnostic. During application design, it is still undefined which products will be coupled to the application. This will occur at deployment time and depends on the preference of the IoT ecosystem owner. Some users or companies are willing to integrate very durable but expensive technologies. Others prefer cheaper solutions.

The application developer creates an XML file that contains all devices used in the application. Device properties on which the developer relies for a proper functioning of the application are included in this file as `QoSRequirements`. Each `QoSRequirement` tag contains the name of the property together with one or more values and units when applicable. Furthermore, the developer defines the criticality of each quality property. This can either be *required* or *desired. required* means that the application will not function properly if the QoS property is not met. Failing to meet *desired* requirements can result in service level degradation. Lastly, the developer can assign a purpose to each QoS requirement, and annotate the specific service level degradation for desired requirements. Including this information can result in meaningful feedback towards infrastructure managers. The listing below shows QoS constraints that can be expressed by app developers in the smart office application.

```
<DeviceSet>
  <Lamp name="meetingRoomLamp">
    <QoSRequirement>
      <importance> required </importance>
      <property> max_response_delay </property>
      <value> 2 </value>
      <unit>sec</unit>
      <purpose>bigger delays can cause
      nightly safety hazards</purpose>
    </QoSRequirement>
    <QoSRequirement>
      <importance> desired </importance>
      <property> change_brightness </property>
      <purpose>changing room brightness</purpose>
      <degradation>no brightness modifications
      </degradation>
    </QoSRequirement>
  </Lamp>
```

```
    <LightSensor name="meetingRoomLightSensor">
        <QoSRequirement>
            <importance> required </importance>
            <property> minimal_polling_frequency </property>
            <value> 2 </value>
            <unit> /hour </unit>
            <purpose> historical data </purpose>
        </QoSRequirement>
        <QoSRequirement>
            <importance> desired </importance>
            <property> minimal_polling_frequency </property>
            <value> 1 </value>
            <unit> /minute </unit>
            <purpose> automatic brightness control</purpose>
            <degradation> no automatic brightness control
            </degradation>
        </QoSRequirement>
    </LightSensor>
</DeviceSet>
```

D. Device selection and coupling

The final step is executed during deployment by the infrastructure provider. The infrastructure provider is responsible for buying, installing and maintaining sensors and actuators. Both are kept in an inventory. The infrastructure provider relies on the XML-file delivered by the application developer for selecting the devices. An engine supports the application developer when selecting feasible devices. The following functionality is supported:

- **Suggestions.** The software tool returns a list of feasible products based on the QoS requirements defined in the XML file provided by the application developer in combination with product properties in the catalogue.
- **Device Comparison.** It is possible to compare the properties of multiple products that meet the imposed requirements.
- **Conflict and degradation feedback.** The infrastructure provider receives feedback about the scope and impact of the application degradation for products that do not meet *desirable* requirements. Clear warnings are shown in case the infrastructure provider selects devices that do not meet *required* properties.

VI. Discussion

In the previous section, the approach was applied to a straightforward case study in which QoS properties are immutable and, hence, can be determined at design time. However, IoT devices can have *configurable properties*. Moreover, increasing the quality level of a given property may negatively impact other quality properties. For example, the monitoring frequency of a sensor can be modified at configuration time. However, this may negatively impact the device's battery life. Similarly, increasing frequency and accuracy parameters may negatively impact bandwidth. During device catalog creation, instead of assigning a static value to a QoS parameter, a relation between the values of multiple parameters or supported intervals for each quality parameter can be expressed. In the device selection step, the engine can return acceptable technologies together with feasible configurations given the QoS requirements that were raised by the application developers. Also, if various technologies and/or configurations are possible, the most optimal configuration of each technology with relation to other quality properties (such as battery life) can be returned.

The case study focused on functional (e.g. `change_brightness` and `change_color`) and reliability (e.g. `delays` and `accuracy`) QoS properties. Many application domains – such as health care and smart home – raise *security and privacy related constraints*. Security and privacy properties can be handled similarly to other QoS properties. Security requirements can by expressed in terms of appropriate authentication (e.g. password versus PKI based) and communication (integrity, confidentiality, non-repudiation...) properties, and/or (un)acceptable underlying security technologies (such as (un)acceptable VPN or TLS versions). Privacy requirements constrain acceptable trust in entities and platforms that are responsible for storing and processing (sensitive) data. The privacy properties assigned to a particular technology then define the set of (sensitive) data that is processed and stored, and the entities that can access raw and aggregated data when that technology is integrated in the IoT app. For instance, some edge technologies can only be monitored and/or controlled by apps in the local network. Others rely on a cloud platform. Each strategy has advantages and constraints. Cloud sensors and actuators are remotely controllable but probably expose more information to third parties. Others may offer improved privacy properties and response times at the cost of decreased accessibility. In home environments, end users may require that no information about light actuation is leaked to third parties for sake of privacy[2]. Some lighting solutions such as Philips Hue support both local and cloud configuration. The specific configuration selected by the infrastructure provider should be compliant with the expressed privacy requirements.

Until now, we assumed that QoS constraints are expressed at development time and enforced at deployment time. Only devices that comply with the requirements are loaded and used in the application. However, at runtime binding and, hence, at runtime QoS enforcement can be necessary in certain application scenarios. Assume the increasing amount of intelligent sensors (air quality sensors, info beacons...) that are currently rolled out by many organizations and even individuals in emerging smart cities. An end-user app typically wants to bind only with smart sensors in his direct surroundings. Security, privacy and accuracy requirements may constrain at runtime coupling. QoS enforcement consists of two phases, namely (1) checking the QoS requirements and (2) tackling conflicts. In the first phase, the IoT device passes its QoS properties to the application. In untrusted environments, the QoS properties can be included in a QoS seal signed by a third party to convince the app. The app inspects the trustworthiness of

[2]Research has already shown that religious background can be derived from the time smart actuators are controlled by residents during certain periods [21].

the properties, and subsequently compares them to the QoS requirements imposed by the app developer. Alternatively, some QoS parameters can be inspected continuously at run time. For instance, an application can monitor the polling frequency of each connected sensor. In the second phase, conflicts are detected and resolved. The conflict resolution strategy depends on the conflict type and application. If a *required* property cannot be met, some app functionality is (temporarely) disabled. When a *desirable* property is not met, it is often possible to perform a graceful degradation of the application's behaviour.

Note finally that we assume that app developers manually define all relevant QoS properties. However, many functional properties can often be extracted automatically from the app code. Some method invocations called by app developers on virtual devices can directly be mapped to required QoS properties. For instance, if an application developer calls the `adjust_brightness()` method, `change_brightness` automatically becomes a quality property that is required when a particular product or technology is selected. To ease the elicitation of required QoS properties, rules can be defined that map method invocations to QoS properties.

VII. CONCLUSION

Device virtualization facilitates technology-agnostic development of IoT applications. This means that app developers can fully focus on implementing business logic. However, IoT applications often impose QoS to underlying sensor and actuator technology either to ensure an acceptable user experience or proper functional behaviour. This paper presents a practical approach to incorporate QoS during the design of IoT applications. Application developers need to take QoS into consideration from the early design stage. Annotating virtual IoT devices with QoS constraints during application development allows infrastructure providers to select the right sensors and actuators during deployment. An IoT device catalogue and compliance engine can support infrastructure providers during the QoS conflict resolution and device selection. We demonstrated the applicability of our approach by means of a smart office scenario.

REFERENCES

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645 – 1660, 2013.

[2] G. White, V. Nallur, and S. Clarke, "Quality of service approaches in iot: A systematic mapping," *Journal of Systems and Software*, vol. 132, pp. 186 – 203, 2017.

[3] S. Karagiorgou, G. Stamoulis, and P. Kikiras, "Enabling qos in the internet of things," 01 2012.

[4] I. Awan and M. Younas, "Towards qos in internet of things for delay sensitive information," in *Trends in Mobile Web Information Systems* (M. Matera and G. Rossi, eds.), (Cham), pp. 86–94, Springer International Publishing, 2013.

[5] A. Malik, J. Qadir, B. Ahmad, K.-L. A. Yau, and U. Ullah, "Qos in ieee 802.11-based wireless networks: A contemporary review," *Journal of Network and Computer Applications*, vol. 55, pp. 24 – 46, 2015.

[6] N. Benamar, A. Jara, L. Ladid, and D. E. Ouadghiri, "Challenges of the internet of things: Ipv6 and network management," in *2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 328–333, July 2014.

[7] K. Govindan and A. P. Azad, "End-to-end service assurance in iot mqtt-sn," in *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, pp. 290–296, Jan 2015.

[8] M. P. Christiansen, S. K. Garg, R. Brazg, B. W. Bode, T. S. Bailey, R. H. Slover, A. Sullivan, S. Huang, J. Shin, S. W. Lee, and F. R. Kaufman, "Accuracy of a fourth-generation subcutaneous continuous glucose sensor," *Diabetes Technology Therapeutics*, vol. 19, pp. 446–456, August 2017.

[9] D. Zhang, H. Chang, P. Li, R. Liu, and Q. Xue, "Fabrication and characterization of an ultrasensitive humidity sensor based on metal oxide/graphene hybrid nanocomposite," *Sensors and Actuators B: Chemical*, vol. 225, pp. 233 – 240, 2016.

[10] W. R, B. G, D. M, and et al, "Accuracy of wrist-worn heart rate monitors," *JAMA Cardiology*, vol. 2, no. 1, pp. 104–106, 2017.

[11] R. K. Behera, K. H. K. Reddy, and D. S. Roy, "Modeling and assessing reliability of service-oriented internet of things," *International Journal of Computers and Applications*, vol. 0, no. 0, pp. 1–12, 2018.

[12] I. Corredor, J. F. Martínez, M. S. Familiar, and L. López, "Knowledge-aware and service-oriented middleware for deploying pervasive services," *Journal of Network and Computer Applications*, vol. 35, no. 2, pp. 562–576, 2012.

[13] N. Small, S. Akkermans, W. Joosen, and D. Hughes, "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, Oct 2017.

[14] D. F. L. Filho and J. R. Amazonas, "Tcnet: Trellis coded network implementation of qos-aware routing protocols in wsns," in *2012 IEEE Latin-America Conference on Communications*, pp. 1–6, Nov 2012.

[15] W. Li, F. C. Delicato, P. F. Pires, Y. C. Lee, A. Y. Zomaya, C. Miceli, and L. Pirmez, "Efficient allocation of resources in multiple heterogeneous wireless sensor networks," *Journal of Parallel and Distributed Computing*, vol. 74, no. 1, pp. 1775 – 1788, 2014.

[16] N. Matthys, F. Yang, W. Daniels, S. Michiels, W. Joosen, D. Hughes, and T. Watteyne, "??pnp-mesh: The plug-and-play mesh network for the internet of things," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pp. 311–315, Dec 2015.

[17] T. Gomes, J. Brito, H. Abreu, H. Gomes, and J. Cabral, "Greenmon: An efficient wireless sensor network monitoring solution for greenhouses," in *2015 IEEE International Conference on Industrial Technology (ICIT)*, pp. 2192–2197, March 2015.

[18] Y. Chen, W. Shen, H. Huo, and Y. Xu, "A smart gateway for health care system using wireless sensor network," in *2010 Fourth International Conference on Sensor Technologies and Applications*, pp. 545–550, July 2010.

[19] C. Abreu, F. Miranda, and P. Mendes, "Smart context-aware qos-based admission control for biomedical wireless sensor networks," *Journal of Network and Computer Applications*, vol. 88, pp. 134 – 145, 2017.

[20] M. Willocx, I. Bohé, J. Vossaert, and V. Naessens, "Developing maintainable application-centric iot ecosystems," in *2018 IEEE International Congress on Internet of Things (ICIOT)*, vol. 00, pp. 25–32, Jul 2018.

[21] J. Bugeja, A. Jacobsson, and P. Davidsson, "On privacy and security challenges in smart connected homes," in *2016 European Intelligence and Security Informatics Conference (EISIC)*, pp. 172–175, Aug 2016.