QoS-based Formation of Software Architectures in the Internet of Things

Martina De Sanctis[1], Romina Spalazzese[2], Catia Trubiani[1]

[1] Gran Sasso Science Institute, L'Aquila, Italy,
{martina.desanctis,catia.trubiani}@gssi.it
[2] Department of Computer Science and Media Technology,
Internet of Things and People Research Center, Malmö University, Sweden
{romina.spalazzese}@mau.se

Abstract. Architecting Internet of Things (IoT) systems is very challenging due to the heterogeneity of connected objects and devices, and their dynamic variabilities such as mobility and availability. The complexity of this scenario is exacerbated when considering Quality-of-Service (QoS) constraints. Indeed, reasoning about multiple quality attributes, e.g., power consumption and response time, makes the management of IoT systems even more difficult since it is necessary to jointly evaluate multiple system characteristics. The focus of this paper is on modelling and analysing QoS-related characteristics in IoT architectures. To this end, we leverage on the concept of Emergent Architectures (EAs), i.e., a set of things temporarily cooperating to achieve a given goal, by intertwining EAs with QoS-related constraints. Our approach provides the automated formation of the *most suitable EAs* by means of a QoS-based optimisation problem. We developed an IoT case study and experimental results demonstrate the effectiveness of the proposed approach.

1 Introduction

The Internet of Things (IoT) refers to a complex network of interactive *things*, i.e., heterogeneous tags, sensors, actuators, objects, and devices that dynamically cooperate [1–3]. The IoT is exploited for the development of many applications spanning multiple domains, such as natural disasters, industrial automation, smart homes [4]. The IoT attracted the attention of companies, governments, and citizens, and has given rise to research in both industry and academia [5]. A recent estimation of the IoT market in the upcoming years has been quantified of being $1.7 trillion including nearly 50 billion things [6].

Nonetheless, building software architectures that support the execution of IoT systems brings new challenges, in fact non trivial choices are required when heterogeneous objects and devices must dynamically cooperate. The IoT environment changes dynamically, e.g., due to devices' availability or the user's mobility. Given the uncertainty in the operational environment (e.g., faulty things, capabilities appearing/disappearing at any moment), and the high diversity of things dynamically available in different and often unknown places, it is not feasible to

define a priori a unique software architecture. Moreover, communicating things may be also potentially resource-constrained. The peculiarity of the IoT domain is that services may show QoS-based characteristics that are platform-specific (e.g., the sensing of light level may be offered by multiple sensor devices, each of them showing a different QoS) and time-varying (e.g., actuators may be constrained by the battery level that changes at runtime), and this heterogeneity makes more complex the QoS-based evaluation of IoT software architectures. This paves the way for considering Quality-of-Service (QoS) concerns in IoT as first class citizens.

In the literature, several QoS-based methodologies have been proposed at various layers of the IoT architecture and different QoS factors, such as performance and reliability, have been considered [7]. However, there is still need for models, metrics, and tools that facilitate the interaction with the dynamically available things, thus to satisfy QoS-related goals, besides the functional ones. This paper focuses on the challenge of specifying IoT architectural models including QoS aspects and providing support for the *automatic* formation of the Emergent Architectures (EAs). EAs stem from Emergent Configurations (ECs), i.e., a set of things that connect and cooperate temporarily through their functionalities, applications, and services, to achieve a user goal [8, 9]. Things are possibly smart connected objects (e.g., curtains) and devices (e.g., temperature sensors). More specifically, we are interested to derive the *most suitable* EAs, i.e., an *optimal set* of connected things cooperating to jointly address functional and extra-functional requirements.

In our previous work [10], we make use of Domain Objects (DOs), i.e., a service-based formalism [11], for forming and enacting ECs in the IoT domain. However, despite its proved effectiveness in the dynamic and automatic formation of ECs, we experienced improper usage of resources, even reflecting to end-users unsatisfaction. To tackle these issues, in this paper we extend both the DOs formalism and the approach in [10] where purely functional requirements can be specified, and QoS-related concerns were not considered at all. The specific contributions of this paper are: (i) a model-based approach that embeds the specification of QoS-related properties at the level of things; (ii) the automated formation of the most suitable EAs in the IoT relying on the selection of QoS-based optimal devices; (iii) a case study demonstrating the feasibility and effectiveness of the proposed approach.

The remainder of this paper is organised as follows. Section 2 describes an IoT scenario that we use throughout the paper, and some background information. Section 3 illustrates our approach. Section 4 presents the case study, explains experimental results, and discusses threats to validity. Section 5 reports related work, and Section 6 concludes the paper pointing out future research directions.

2 Motivating example and foundations

In this section we give a motivating scenario that will guide us through the paper and we describe some relevant background notions.

2.1 Smart Light scenario

In this section we describe the IoT Smart Light (SL) scenario, where things cooperate to achieve a predefined light level in a lecture room. This scenario extends the one described in [10] by further including and managing extra-functional requirements. Consider, for instance, a university campus made by different buildings hosting diverse types of rooms, e.g., libraries, dormitories, classrooms, offices. Each room is equipped with several IoT things, i.e., light sensors, curtains, and lamps. The things, along with their functionalities, are configured to be controllable via a mobile application allowing authorized users to increase/decrease the light level while moving in different rooms, based on their needs. For instance, in a lecture room, the lecturer can decide to decrease the light level when giving a presentation through a projector or, to the contrary, to increase it when using the blackboard. As opposite, in a dormitory room, a student can decide to have a higher light level when studying and a lower one when resting. A possible way to achieve such goals is to dynamically identify an EA made, for instance, by the user's smartphone, a light sensor, and available curtain(s) and lamp(s). The selected light sensor measures the current light level in the room, and subsequently the lamps are turned on/off, and the curtains can be opened or closed.

Besides fulfilling the functional goals of this scenario (e.g., adjusting the light level), the mobile application committer and the final users are also interested in fulfilling extra-functional requirements. For instance, the committer may want to minimise the power consumption of all the devices installed in the campus, i.e., to positively impact on the campus energy bill, while guaranteeing users satisfaction. This means that users can set their own preferences modifying the default settings. Specifically: (i) light sensors display different sensing accuracy and users may require a certain accuracy level to get trustable estimations; (ii) curtains expose a time required for opening/closing them, and users may be interested in minimising it; (iii) lamps contribute with different light intensities, and users may select the ones that better match with the required light level.

2.2 Background

This work builds upon an existing approach called IoT-FED (Forming and enacting Emergent configurations through Domain objects in the IoT) [10] that exploits the Domain Object (DO) model, i.e., the building block of a design for adaptation [11].

Domain Objects. DOs allow the definition of independent and heterogeneous things/services in a uniform way. This means that developers can work at an abstract level without dealing with the heterogeneity of things and their communication protocols. Since the actual system can vary in different execution contexts, as it can be constituted by disparate things (e.g., sensors, actuators, smartphones) dynamically available, the DO model supports the systems realization at runtime, when the execution context is known.

To model things, developers wrap them as DOs. This task is done only *una tantum*, i.e., when a new device type/brand is available. Each DO implements its own behavior (i.e., the *core process*), which is meant to model its capability (e.g., the sensing capability of a sensor). At the same time, for its nominal execution, a DO can optionally require capabilities provided by other DOs (e.g., the lamp and curtain actuating capabilities are externally required by the SL application). In addition, it exposes one or more *fragments* (e.g., the sense light level fragment) describing offered services that can be *dynamically* discovered and used by other DOs. Both core process and fragments are modelled as processes, by means of the Adaptive Pervasive Flows Language (APFL) [12].

The dynamic cooperation among DOs is performed by exploiting a *refinement* mechanism. At design time, APFL allows the partial specification of the expected behaviour of a DO through *abstract activities*, i.e., activities that the DO does not implement itself; they are defined only in terms of a goal labelling them (e.g., sense the light) and they represent open points in DOs' processes and fragments. At runtime, the refinement mechanism makes abstract activities *refined* according to the (composition of) fragments offered by other DOs, whose execution leads to achieve the abstract activity's goal. This enables a *chain of refinements*, as will be later discussed (see Figure 5). We adopt a refinement mechanism that makes use of advanced techniques for the dynamic and incremental service composition, and it is based on Artificial Intelligence (AI) planning [13]. For further details on DOs, we refer to [14] describing the prototype of a travel assistant application developed by using DOs technologies.

IoT-FED. The IoT-FED approach supports the formation and enactment of ECs by means of the DOs technologies. Given the *user goal type* (e.g., adjust light level) and the *goal spatial boundaries*, such as the location where the EC must be formed and enacted (e.g., the lecture room), the execution starts (e.g., from the SL application's DO). If existing, the EC is made up by the set of things whose corresponding DOs have been involved in the refinement process of all the encountered abstract activities, through the selection of their fragments.

Figure 1 shows an abstract framework [10] where the shaded box highlights the newly defined component for QoS-related concerns, whereas the boxes with the striped pattern highlight the components that have been modified to handle QoS aspects. In the following we describe the main components.

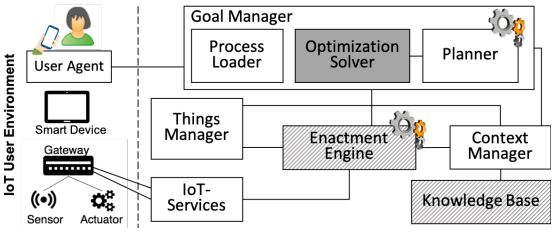

Fig. 1: Overview of our framework.

The *Goal Manager* is responsible for parsing the user goal and starting the EC formation process. It has three sub-components: (i) the Process Loader, responsible for specifying the user goal type and the spatial boundaries, and for loading the DO process corresponding to the specified goal type; (ii) the Opti-

mization Solver (that will be detailed in Section 3.1); (iii) the Planner responsible for the refinement of abstract activities in the loaded process. The *Things Manager* is responsible for managing available IoT things and DOs. It answers queries about available IoT things, their capabilities and locations; dynamically instantiates needed DOs, and handles co-relations among them. The *Enactment Engine* is mainly responsible for enacting the ECs. It (i) forms and enacts the ECs; (ii) sends instructions to IoT things (e.g., get sensor readings) through the Things Manager; (iii) handles the injection of the plans received by the Planner in place of the abstract activities and (iv) executes the final refined process that achieves the user goal. The *Context Manager* is responsible for maintaining the system knowledge. It retrieves data from the knowledge base (KB), parses received context from the Enactment Engine (e.g., new things states), and updates the KB. The *Knowledge Base* holds the internal system knowledge and includes repositories storing things operational states (e.g., if lights are turned on or off), the designed DOs, and the associations among things, DOs and corresponding capabilities. The *IoT Services* component enables the management and interaction with things, and it relies on the Amazon AWS-IoT cloud platform[3].

To make IoT things and services available in IoT-FED, a developer needs to do two main operations: (i) *register things in the AWS-IoT platform;* (ii) *model things, services and applications as DOs.* The REST endpoints generated by the platform are invoked in the DOs processes activities.

3 QoS-based approach

Our approach provides mechanisms for determining the near-optimal IoT-EAs that jointly satisfy functional and extra-functional requirements. In this section we use the SL scenario described in Section 2.1, where things cooperate for reaching the goal to set a predefined light level in a lecture room.

3.1 Overview of the approach

This section describes the extensions made to the IoT-FED approach to enable the automatic QoS-based formation of EAs. To allow developers to specify QoS-related characteristics of things, we extended the Domain Objects formalism. This extension clearly impacts on the modelling phase of domain objects (see the shaded box in Figure 2).

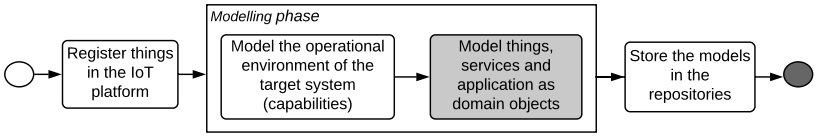

Fig. 2: IoT-FED extended guideline.

The specification of QoS-related characteristics, indeed, is performed at the level of DOs. In particular, each thing is associated to an arbitrary number

[3] `https://aws.amazon.com/it/iot`

of metrics inherited from its producer. Thus, we enhanced the specification of DOs (e.g., those representing real world things in the environment, value-added services or user applications) by adding QoS-related attributes.

Figure 3 reports an example of a domain-specific sensor (i.e., the Sensmitter[4]) expressed as a .xml file representing the corresponding light sensor's DO. In particular, it shows that the DO's state also contains QoS-related attributes (see lines **16-24** of Figure 3), besides state variables. Specifically, regarding the

```
1<?xml version="1.0" encoding="UTF-8"?>
2<tns:domainObject name="SensmitterLightSensor" xmlns:tns="http://.../">
3
4    <tns:domainKnowledge>
5        <tns:internalDomainProperty name="domainProperties/LightSensing">
6    </tns:internalDomainProperty>
7    </tns:domainKnowledge>
8    <!-- List of state variables -->
9    <tns:state>
10       <tns:stateVariable name="DeviceID" type="string">
11           <tns2:content type="anyType">Sensmitter_435</tns2:content>
12       </tns:stateVariable>
13       <!-- Other state variables here -->
14
15       <!-- QoS-related attributes -->
16       <tns:QoSAttribute name="PowerConsumption" type="integer">
17           <tns2:content type="anyType">2.5</tns2:content>
18       </tns:QoSAttribute>
19       <tns:QoSAttribute name="SensingAccuracy" type="integer">
20           <tns2:content type="anyType">8</tns2:content>
21       </tns:QoSAttribute>
22       <tns:QoSAttribute name="BatteryLevel" type="integer">
23           <tns2:content type="anyType">100</tns2:content>
24       </tns:QoSAttribute>
25   </tns:state>
26
27   <tns:process name="processes/PROC_SensmitterLightSensor"/>
28   <tns:fragment name="fragments/LS_senseLight"></tns:fragment>
29
30</tns:domainObject>
```

Fig. 3: Domain object model for the Sensmitter light sensor.

SL scenario with the three categories of used devices, the specification of light sensors is augmented with three metrics: (i) power consumption (see lines **16-18** of Figure 3), i.e., the energy consumed by sensors to provide measurements on the light level; (ii) sensing accuracy (see lines **19-21** of Figure 3), i.e., the precision provided by sensors about their estimations; (iii) battery level (see lines **22-24** of Figure 3), i.e., the state of the device's battery that can be dynamically updated. Lamps and curtains also include the power consumption in their specification, but differently from sensors, lamps show a lighting level that expresses their intensities, and curtains show a timing for opening/closing that denotes the efficiency of such devices. Note that metrics can be expressed in different units for sensors and actuators of different brands, however such units can be converted to a common reference unit in the DO model, thus to avoid misleading comparison.

The default setting of extra-functional requirements (i.e., min, max, threshold value) is enabled by the developers in the setting of the SL application. However,

[4] `https://www.senssolutions.se/`

end-users may have different preferences while using the available things, hence they can modify such requirements. This is later translated into the QoS-based optimisation problem that guides the formation of the most suitable EAs.

The aim of our approach is to verify if an EA can be formed to achieve the given (functional and extra-functional) goal in the specified spatial boundaries. In particular, this is strictly related to the refinement of abstract activities. We recall that the refinement process consists of the automated resolution of a fragments composition problem. It is transformed into a planning problem, and AI planning-based techniques are used to solve it.

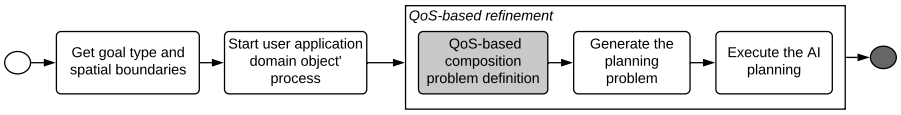

Fig. 4: IoT-FED extended process.

In particular, we enhanced the fragments composition problem in such a way that it also considers the QoS-related characteristics of devices. In Figure 4 we provide an abstraction of the IoT-FED extended process (see the shaded box in Figure 4). This way, the generated planning problem considers both extra-functional requirements and QoS-based characteristics expressed by DOs. Indeed, the specification of QoS-related characteristics in the DOs, together with the setting of extra-functional requirements (i.e., min, max, threshold value), leads to multiple architectural alternatives and trade-off analyses for the selection of near-optimal EAs. The mentioned QoS-based optimization problem is defined and solved by the **Optimization Solver** component (Figure 1).

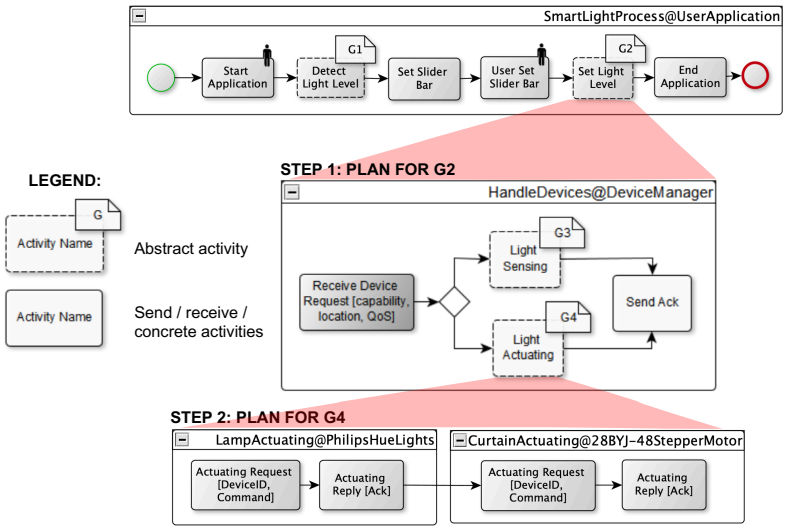

Fig. 5: Smart Light execution example.

Figure 5 depicts a simplified example of the SL application execution. The Smart Light Process denotes the specification of the user application, and it

represents the *User Application* DO. The QoS-based requirements guide the refinement of the encountered abstract activities (i.e., *Detect Light Level*, *Set Light Level*). For instance, the refinement of the *Set Light Level* abstract activity (i.e., goal G2 in Figure 5) includes the fragment *Handle Devices* that is provided by the *Device Manager* DO (see Step 1 of Figure 5). If the selected DO is not instantiated, then such operation is performed by the Things Manager component. The execution of this fragment implies the co-relation between the two instantiated DOs (i.e., *User Application* and *Device Manager*). The settled extra-functional requirements are passed to the Device Manager, see the QoS input data in the *Receive Device Request* activity. Subsequently, it will be considered for the refinement of the *Light Actuating* (i.e., goal G4 in Figure 5) abstract activity. Eventually, the fragments composition (returned for this last refinement) is made by two fragments provided by those actuators in the room whose QoS-related characteristics are compliant with the QoS-based optimisation problem (see Step 2 of Figure 5). Specifically, the fragments *Lamp* and *Curtain Actuating*, respectively provided by the *Philips Hue Lights*[5] and the *Stepper Motor*[6] DOs are selected, composed and injected in place of the abstract activity they refine.

3.2 Deriving QoS-based optimal IoT-EAs

The QoS-based search for alternative EAs initially deals with the issue of finding a set of devices (D) that implement the functionalities required by the application, but also fulfilling the stated requirements. Note that considering the requirements leads to trade-off analysis that takes into account the dependencies among the QoS-based properties, thus reducing the solution space. Then, given this reduced solution space (i.e., a set of devices), it is the planner to look for the optimal EAs. More formally, the problem can be expressed as follows: *Look for an EA_{opt} derived by an optimal selection of devices D_{opt}.*

To this extent, we defined an optimisation problem. It may also be modified by adding some constraints, such as costs and further domain-specific characteristics, e.g., the charging level of available devices. In the constrained case, we assume that there is a cost/restriction associated with each EA for providing a certain QoS level. In our case study, for example, we can introduce a constraint regulating the charging level of devices before being selected, e.g., activate lamps or curtains showing an initial battery level larger than 80%.

The number of different EAs (generated from the D_{opt}) is conditioned to the number of sensors and actuators, plus their instances. Since sensors are selected before looking for actuators, we firstly need to evaluate the search space for sensors. This is $\mathcal{O}(\sum_{i=1}^{n} s_i)$ where n is the number of sensor types, and s_i is the number of sensor instances (related to the $i-th$ sensor type) that can be used to form any EA and contributing to provide a specific sensing service (e.g., light).

The number of possible EAs also depends from the number of actuators and their instances. For each actuator type a_j (with $j = \{1, \ldots, k\}$) we get a complexity of $\mathcal{O}(\sum_{i=1}^{m_j} a_{ji})$ where m_j is the number of the $j-th$ actuator type, and

[5] `https://www2.meethue.com/en-us` [6] `https://bit.ly/2VmRegr`

a_{ji} is the number of actuator instances for the corresponding $j-th$ type, that can be used to form any EA and contributing to provide a specific actuating service. All actuator types and their instances contribute to the search space. When actuators are individually selected, the complexity is $\mathcal{O}(\sum_{j=1}^{k}\sum_{i=1}^{m_j} a_{ji})$, whereas their combination is given by $\mathcal{O}(\prod_{j=1}^{k}\sum_{i=1}^{m_j} a_{ji})$. We recall that a_j represents the $j-th$ actuator type and the complexity of its instances is inherited from the definition above. Thus, the size of the solution space for the optimisation problem is $\mathcal{O}(\sum_{i=1}^{n} s_i \times \sum_{j=1}^{k}\sum_{i=1}^{m_j} a_{ji} \times \prod_{j=1}^{k}\sum_{i=1}^{m_j} a_{ji})$, and it becomes clear that it may be huge even for small values of s_i, and a_j. For example, in our case study, we considered the following setting: $s_i = 5$, $n_1 = \ldots = n_5 = 20$, $a_j = 2$, $m_1 = 3$, $m_2 = 2$, and the size of the solution space is $100 * 5 * 6 \simeq 3$k options which makes an exhaustive search computationally expensive.

To address this challenge, we describe a near optimal solution technique that takes as input the specification of all the available devices, in the user spatial boundaries, whose fragments (exposed by the corresponding DOs) are suitable for the resolution of the planning problem. Such devices are analysed and discarded from the optimal set whenever their QoS-related characteristics do not fulfil the stated requirements. This set of devices, namely D_{opt}, is provided as output, and it contributes to the domain for the planning.

An optimal selection of devices is performed taking into account the application's settings (also editable by the users). If minimisation or maximisation is required, then an exhaustive search is necessary. On the contrary, if threshold values are set, then it is needed to look for the subset of sensors that fulfil such requirements. This way, the overall set of selected devices for both sensors and actuators is guaranteed to fulfil the stated extra-functional requirements.

4 Experimentation

This section reports our experimental results for the SL case study. We are interested to evaluate the QoS-based characteristics while selecting things (i.e., sensors and actuators). The scalability of the approach is also investigated.

4.1 Experimental setup and results

Table 1 reports the QoS-related characteristics of different brands of light sensors, lamp and curtain actuators, respectively. Our case study includes five brands of light sensors showing a power consumption (pc) varying from 1 to 5 watts[7], and a sensing accuracy (sa) is spanning from 2 to 10 and denoting an increasing precision. These two QoS-related characteristics are complementary, in fact a higher accuracy is given by a larger power consumption. Lamp actuators are of three different brands where power consumption varies between 10 to 20 watts[8], whereas their light level (ll) is spanning from 4 to 8 and it indicates an increasing brightness. As another example, curtains are of two brands with

[7] See, e.g., `https://bit.ly/2IC6jtd` [8] See, e.g., `https://bit.ly/2TibLWj`

an associated power consumption of 7 and 9 watts[9], and a discrete timing for opening/closing (toc) equal to 8 and 12 seconds, respectively. All these numbers represent an estimation of QoS-related characteristics for arbitrary things, however their actual setting is part of the modelling step, and further numerical values can be considered when more accurate specification of things is available.

Table 1: QoS-related characteristics.

	Light Sensors					**Lamps**			**Curtains**	
	LS_1	LS_2	LS_3	LS_4	LS_5	LA_1	LA_2	LA_3	CA_1	CA_2
pc	1	1.8	0.5	5	2.5	10	20	15	7	9
sa	4	7	2	10	8	-	-	-	-	-
ll	-	-	-	-	-	4	8	6	-	-
toc	-	-	-	-	-	-	-	-	12	8

All experimental results are obtained by using a laptop equipped with a dual-core CPU running at 2.7GHz, and 8Gb memory. In the following we discuss three main experiments that have been performed to evaluate different aspects of the SL case study. For all experiments we report the average values calculated over one hundred runs of the SL application. Besides, the execution time of the overall process is showed to demonstrate that its latency is affordable. In fact, we anticipate that all execution times, measured from when the user starts the SL application to the enactment of the QoS-based formed EA, vary up to 2.14 seconds, even when handling up to one hundred devices.

Exp_1: evaluation of QoS-related characteristics for sensors only. This experiment is aimed to understand the savings when adding extra-functional requirements for a specific device type. In our case study, we evaluated what happens when incrementally adding requirements (expressed with threshold values) to the power consumption and sensing accuracy of sensors. Obviously, we achieve a consistent power consumption saving (up to 50%) when considering constraints on it; however, when also including sensing accuracy, we still get 35% of savings that is a remarkable improvement. In this last case, the sensing accuracy increases, as expected, and this is due to the trade-off analysis among these two metrics. More in general, the requirements can be separately considered and lead to optimisation problems that provide different solutions.

Table 2: QoS-based optimisation for sensors.

	noQoS	QoS(pc)	QoS(pc, sa)
Power consumption	233.1	117.7	140.8
Sensing accuracy	6.41	4.63	5.53
Execution time	1.96	1.97	2.14

Table 2 reports the values of Exp_1, and it is structured as follows. Rows include the metrics we are considering to quantify the QoS-based savings, specifically: (i) the power consumption of the sensors used in the EAs; (ii) the sensing accuracy of adopted sensors; (iii) the execution time (expressed in seconds) for forming and enacting EAs. On the columns we distinguish three cases: the first one is without setting any constraint on QoS-related characteristics (i.e., *noQoS* in Table 2), the second is when setting the power consumption

[9] See, e.g., `https://bit.ly/2NYwPKF`

of sensors being less than 2 watts (i.e., *QoS(pc)* in Table 2), and the third case is when also establishing that the sensing accuracy has to be larger than 3 (i.e., *QoS(pc, sa)* in Table 2). We can notice that QoS-based savings are relevant for our case study, in fact power consumption goes from 233 to 118. This implies a modification in the sensing accuracy that instead decreases (from 6.41 to 4.63), due to the selection of light sensors that consume less. However, the value for the sensing accuracy slightly improves to 5.53 when setting the threshold to that metric. Execution times also slightly increase across experiments when adding QoS-related constraints, but the largest gap is equal to 2.14 - 1.96 = 0.18 seconds.

Exp_2: evaluation of QoS-based characteristics for all devices. This experiment investigates the savings when adding extra-functional requirements for both sensors and actuators. In our case study, we evaluated QoS-based savings when adding threshold values to the power consumption of all devices and progressively considering further aspects for light sensors, curtains, and lamps.

Table 3: QoS-based optimisation for sensors and actuators.

	noQoS	QoS_1	QoS_2	QoS_3
Power consumption	1946.5	1792.4	1588	1828
Sensing accuracy	6.37	4.35	4.29	4.45
Execution time	1.92	1.91	1.88	1.91
Lighting level	6.12	6	4.86	6
Time opening/closing	9.85	9.91	8	8

Table 3 reports the values of Exp_2, and it is structured as follows. Similarly to Table 2, the first three rows report power consumption (that is measured taking into account all devices), sensing accuracy, and execution time. Last two rows extend the evaluation to the following metrics: (i) the lighting level of lamp actuators used in the EAs; (ii) the time for opening/-closing curtain actuators involved in the EAs. On the columns we present four different cases. The first one is without QoS-related constraints, i.e., *noQoS* in Table 3. The second (denoted by QoS_1 in Table 3) is a combination of: (1) power consumption of sensors (required to be less than 2 watts), lamp actuators (required to be less than 18 watts), curtain actuators (required to be less than 10 watts); (2) lighting level of lamp actuators (required to be larger than 5). The third case (i.e., QoS_2 in Table 3) keeps the same threshold values for the power consumption, but it requires a minimisation of time for opening/closing curtains. Finally, the forth case (denoted by QoS_3 in Table 3) is a combination of the previous two cases where thresholds for power consumption, lighting level and time for closing/opening are jointly considered.

Obviously, we can notice that in all QoS-based optimisation procedures power consumption shows an improvement with respect to *noQoS*, in fact it goes from 1946.5 up to 1588 in the best case. As drawback, the sensing accuracy decreases and goes from 6.4 to values around an average of 4.3 (that is larger than the stated threshold). Execution times are very similar in all cases, and this supports the efficiency of QoS-based computation. Lighting level varies across cases and achieves its worst value (i.e., 4.86, see Table 3) when not constrained by any threshold. Finally, the time for opening/closing is also subject to some variations that steer it down in cases where such metric is explicitly optimised, i.e., in the

last two columns of Table 3 where it shows a value of 8 vs the initial 9.85 (i.e., when measured with no QoS-based constraints). Figure 6 depicts the number of alternative EAs. We can notice that in case of not considering QoS-based requirements there are almost 50 different EAs that are enacted.

As expected, the handling of QoS-related requirements implies a reduction in the number of valid EAs of roughly 60%, in fact the average value of EAs in QoS scenarios is around 12. Obviously, QoS_3 is the one showing the lowest value, since it represents a combination of requirements set for QoS_1 and QoS_2. Power consumption (see Table 3) is reported on top of bars in Figure 6 to remark the variations in QoS-based savings.

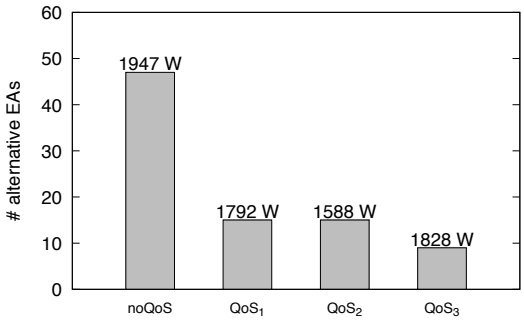

Fig. 6: Variations in the selection of EAs.

Exp_3: scalability of the approach. We added ten and twenty light sensors instances for each of the considered sensor brand, thus to evaluate the scalability of the approach when considering up to 50 and 100 light sensors, respectively.

Table 4: Scalability of the approach.

	noQoS	QoS(pc)	QoS(pc, sa)
#sensors= 5	1.96	1.97	2.14
#sensors= 50	2.01	1.98	1.99
#sensors= 100	2.10	2.05	2.07

Table 4 shows the execution times (expressed in seconds) when varying the number of sensors on the rows, and the cases of Exp_1 in the columns. We found that in all cases the execution time values vary within a narrow interval, i.e., from 1.96 to 2.14 seconds. This supports the scalability of our approach since the QoS-based computation does not largely affect the process of forming and enacting EAs. The number of devices does not affect the scalability of the approach, since the planner component of our platform (see Figure 1) only considers the device brands and it is not checking their instances when computing plans (i.e., fragments compositions).

Summarising, these three experiments provide a quantitative evaluation of our approach and point out two main findings: (i) the effectiveness, since both Exp_1 and Exp_2 clearly show QoS-based savings; (ii) the scalability, in fact Exp_3 indicates that in the worst case the application of the approach takes 2.14 seconds, and this can be considered affordable for those classes of IoT systems that do not expose safety or hard real time constraints.

4.2 Discussion

Our approach includes a set of limitations that we discuss in the following.

Runtime monitoring. There are some QoS-related characteristics associated to things that may change over time, e.g., the battery level of devices decreases

when they are in use or increases after charging. These aspects of runtime evolution of things are currently not handled by our approach that instead computes some preliminary check on the current status of devices only. However, we plan to update these changing values and trigger a QoS-based adaptation (e.g., switching among actuators showing similar QoS-based characteristics but with different battery level) of EAs periodically.

Spatial boundaries. Our analysis is performed taking into account the user spatial boundaries (e.g., a room). In principle, the number of things can scale with order of magnitude larger than the ones considered in this paper. However, here we experimented our QoS-based approach by varying the number of sensor instances (in the lecture room of our case study) up to one hundred. We leave as future work the evaluation of scaling the number of actuators, but we expect that this does not affect too much our computation due to the intrinsic nature of the planner component that reasons on device types (brands) instead of instances.

Requirements specification. It may happen that the QoS-based optimisation problem is not able to provide a solution due to invalid requirements. However, there might be some options that do not deviate largely from users' expectations, and we plan to provide these alternatives as feedback to users that may decide to change their initial settings. Moreover, we plan to introduce weights associated to users' preferences on the type of available things. For example, in the SL application it may happen that users get disturbed by the curtains opening and closing, hence the activation of lamps is preferred.

Further architectural layers. Our approach currently allows the specification of QoS-related characteristics for the sensors and actuators only, because they are the main components for building the EAs. However, between these two layers there are further architectural layers, such as different middleware, operating systems, and communication protocols that contribute to the QoS of the IoT system. As future work we plan to extend our approach to embed these layers in the specification of QoS-related characteristics. An option can be to integrate benchmarks modelling the delay of these layers in our optimisation problem.

Threats to validity. Our experimentation may be internally biased from the settings of input parameters, QoS requirements, and executions of the SL application. Both input parameters and QoS requirements lead to specify different QoS optimisation problems, but the overall procedure is not affected. As opposite, we found relevant to execute multiple runs of the SL application and we experienced no variations between 50 and 100 runs, hence this latest number has been considered to smooth biases in the output results. As external threats to validity, we are aware that the application of the approach to other case studies has not been performed, but we leave this point as part of our future work.

5 Related work

The work presented in this paper is related to two main streams of research that we discuss hereafter, specifically the modelling of IoT architectures and their QoS-based analysis.

In [5] a reference architecture to plug and produce industrial IoT systems (whose architectural decisions are tackled in [15]) is presented, and it has the goal to reduce industrial device commissioning times across vendor products. Differently from our approach, the evaluation of IoT architectures in [5, 15] builds upon some industrial (communication protocol and controller description) standards. On the contrary, our formalism (based on DOs) is aimed to specify any QoS-related characteristic, by setting QoS-based criteria, such as minimisation, maximisation, or a specific threshold value.

In [16] a framework for self-architecting service-oriented systems is proposed, and QoS-based analysis is performed by quantifying the execution time and availability of the service providers. In [17] QoS-based optimisation of service-based systems is performed through modelling the application with a Discrete Time Markov Chain (DTMC) and using a probabilistic model checker to rank the configurations based on the required extra-functional requirements. In [18] an approach for QoS-based feedback on service compositions is presented, and it makes use of design-time and runtime knowledge to manage QoS data over time [19], thus to support software architects while devising a service composition that best fits extra-functional requirements. Our approach mainly differs from these works [16–19] in considering the issues of the IoT domain where services may show QoS-based characteristics that are not platform-independent and time-varying. This heterogeneity makes more complex the QoS-based optimisation problem. In [20] models at runtime and statistical techniques are combined to realise adaptation of IoT systems, specifically quality models provides a probabilistic estimate of different adaptation options. Our approach differs in the specification of QoS-related characteristics that are explicitly modelled at the architectural level and contribute to the selection of devices fulfilling functional and extra-functional requirements.

In recent years, research has been done on the usage of business process-based technologies in the IoT context. Indeed, Business Process Management Systems (BPMS) approaches have become an efficient solution for the coordinated management of devices, as reported in [21]. At the same time, interesting research challenges arise from this novel research field [22]. From the one side, workflow management systems (WfMS) for industrial IoT have been realized to execute and monitor IoT-based processes [23]. From another side, standard workflow languages (e.g., BPMN 2.0) have been extended to support sensors/actuators specific activities and IoT communication paradigms [24]. In our approach, the use of the APFL and the abstract activities refinement mechanism enables the dynamic execution of IoT applications. Moreover, APFL has been extended to support the specification of QoS-related characteristics of things, inherited from their producers and enabling QoS-based formation of software architectures. To the contrary, APFL extensions to support things activities and IoT communication paradigms were not necessary. This is due to the use of the DOs formalism that allows developers to work at an abstract level without dealing with the heterogeneity of things and their communication protocols.

Summarising, we can conclude that, to the best of our knowledge, there is no work that incorporates QoS-related characteristics in the modelling of IoT software architectures and exploits this specification to jointly optimise functional and extra-functional requirements.

6 Conclusion

In this paper we presented an approach to consider QoS-related concerns as first class citizens in the process of forming software architectures in the Internet of Things. We extended a modelling language for enabling the specification of QoS-related characteristics of things (tags, sensors, actuators, objects, and devices). This information is exploited in the automatic formation of EAs since a QoS-based optimisation problem is adopted, and devices are selected taking into account extra-functional requirements. The approach is applied to a case study and the conducted experimentation provides three main lessons learned: (i) when introducing extra-functional requirements, the savings may be relevant; (ii) when considering multiple QoS-related characteristics, trade-off analysis is suitable to balance among contradicting QoS-based goals; (iii) the scalability of the approach is preserved when considering a realistic number of devices. As future work, besides addressing the limitations that have been discussed in the experimentation, we also plan to further investigate the effectiveness of our approach when involving real-world things and industrial case studies.

Acknowledgments. This work has been partially supported by the MIUR PRIN project titled "Designing Spatially Distributed Cyber-Physical Systems under Uncertainty (SEDUCE)".

References

1. L. Atzori and A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.
2. 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.
3. I. Lee and K. Lee, "The Internet of Things (IoT): Applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431–440, 2015.
4. R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future internet: the internet of things architecture, possible applications and key challenges," in *International Conference on Frontiers of Information Technology (FIT)*, 2012, pp. 257–260.
5. H. Koziolek, A. Burger, and J. Doppelhamer, "Self-Commissioning Industrial IoT-Systems in Process Automation: A Reference Architecture," in *International Conference on Software Architecture (ICSA)*, 2018, pp. 196–205.
6. C. MacGillivray, V. Turner, and M. Shirer, "Explosive Internet of Things Spending to Reach $1.7 Trillion in 2020," *IDC Corporate USA*, 2015.
7. G. White, V. Nallur, and S. Clarke, "Quality of service approaches in IoT: A systematic mapping," *Journal of Systems and Software*, pp. 186–203, 2017.

8. F. Alkhabbas, R. Spalazzese, and P. Davidsson, "Architecting Emergent Configurations in the Internet of Things," in *IEEE International Conference on Software Architecture (ICSA)*, 2017, pp. 221–224.
9. F. Ciccozzi and R. Spalazzese, "MDE4IoT: Supporting the Internet of Things with Model-Driven Engineering," in *International Symposium on Intelligent Distributed Computing (IDC)*, 2017, pp. 67–76.
10. F. Alkhabbas, M. De Sanctis, R. Spalazzese, A. Bucchiarone, P. Davidsson, and A. Marconi, "Enacting Emergent Configurations in the IoT Through Domain Objects," in *International Conference on Service-Oriented Computing (ICSOC)*, 2018, pp. 279–294.
11. A. Bucchiarone, M. De Sanctis, A. Marconi, M. Pistore, and P. Traverso, "Design for adaptation of distributed service-based systems," in *International Conference on Service-Oriented Computing (ICSOC)*, 2015, pp. 383–393.
12. A. Bucchiarone, M. De Sanctis, A. Marconi, M. Pistore, and P. Traverso, "Incremental composition for adaptive by-design service based systems," in *International Conference on Web Services (ICWS)*, 2016, pp. 236–243.
13. P. Bertoli, M. Pistore, and P. Traverso, "Automated composition of web services via planning in asynchronous domains," *Artif. Intell.*, vol. 174, pp. 316–361, 2010.
14. A. Bucchiarone, M. De Sanctis, and A. Marconi, "ATLAS: A world-wide travel assistant exploiting service-based adaptive technologies," in *International Conference on Service-Oriented Computing (ICSOC)*, 2017, pp. 561–570.
15. S. Malakuti, T. Goldschmidt, and H. Koziolek, "A Catalogue of Architectural Decisions for Designing IIoT Systems," in *European Conference on Software Architecture (ECSA)*, 2018, pp. 103–111.
16. D. Menasce, H. Gomaa, J. Sousa *et al.*, "Sassy: A framework for self-architecting service-oriented systems," *IEEE software*, vol. 28, no. 6, pp. 78–85, 2011.
17. R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic QoS management and optimization in service-based systems," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 387–409, 2011.
18. M. Caporuscio, R. Mirandola, and C. Trubiani, "QoS-based Feedback for Service Compositions," in *International Conference on Quality of Software Architectures (QoSA)*, 2015, pp. 37–42.
19. R. Mirandola and C. Trubiani, "A Deep Investigation for QoS-based Feedback at Design Time and Runtime," in *International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2012, pp. 147–156.
20. D. Weyns, M. U. Iftikhar, D. Hughes, and N. Matthys, "Applying Architecture-Based Adaptation to Automate the Management of Internet-of-Things," in *European Conference on Software Architecture (ECSA)*, 2018, pp. 49–67.
21. C. Chang, S. N. Srirama, and R. Buyya, "Mobile cloud business process management system for the internet of things: A survey," *ACM Comput. Surv.*, vol. 49, no. 4, pp. 70:1–70:42, 2017.
22. C. Janiesch and et Al., "The internet-of-things meets business process management: Mutual benefits and challenges," *CoRR*, vol. abs/1709.03628, 2017.
23. R. Seiger, S. Huber, and T. Schlegel, "Toward an execution system for self-healing workflows in cyber-physical systems," *Software and System Modeling*, vol. 17, no. 2, pp. 551–572, 2018.
24. D. Domingos, F. Martins, C. Cândido, and R. Martinho, "Internet of things aware WS-BPEL business processes context variables and expected exceptions," *J. UCS*, vol. 20, no. 8, pp. 1109–1129, 2014.