Enacting Emergent Configurations in the IoT through Domain Objects

Fahed Alkhabbas[1,2], Martina De Sanctis[3], Romina Spalazzese[1,2]
Antonio Bucchiarone[3], Paul Davidsson[1,2], Annapaola Marconi[3],

[1] Department of Computer Science and Media Technology, Malmö University, Sweden
[2] Internet of Things and People Research Center, Malmö University, Sweden
{fahed.alkhabbas,paul.davidsson,romina.spalazzese}@mau.se
[3] Fondazione Bruno Kessler, Via Sommarive, 18, Trento, Italy,
{bucchiarone,msanctis,marconi}@fbk.eu

Abstract. The Internet of Things (IoT) pervades more and more aspects of our lives and often involves many types of smart connected objects and devices. User's IoT environment changes dynamically, e.g., due to the mobility of the user and devices. Users can fully benefit from the IoT only when they can effortlessly interact with it. To accomplish this in a dynamic and heterogenous environment, we make use of Emergent Configurations (ECs), which consist of a set of things that connect and cooperate temporarily through their functionalities, applications, and services, to achieve a user goal. In this paper, we: (i) present the IoT-FED architectural approach to enable the automated formation and enactment of ECs. IoT-FED exploits heterogeneous and independently developed things, IoT services, and applications which are modeled as Domain Objects (DOs), a service-based formalism. Additionaly, we (ii) discuss the prototype we developed and the experiments run in our IoT lab, for validation purposes.

1 Introduction

Since the technology becomes more and more affordable and connectivity widespread, most objects and devices that would gain from being connected to the Internet are being connected. We refer to the (possibly smart) devices and connected objects as things [1, 2]. Big market players, e.g., Ericsson[1], foresee that in the coming years, things will form large heterogeneous and highly distributed systems. As a result, the Internet of Things (IoT) will pervade and potentially improve many aspects of our lives.

From a user perspective, the IoT environment changes dynamically, e.g., when the user moves or due to mobile devices. Given the high diversity of things dynamically available in different and often unknown places, it is not feasible to define a priori all the possible combinations of things to reach specific user goals. Additionally, users can fully benefit from the IoT only when they can effortlessly interact with the dynamically available things to satisfy their goals. To accomplish this, a significant engineering effort is needed to abstract from the low-level interactions with the things. To this aim, we

[1] https://www.ericsson.com/res/docs/2015/mobility-report/ericsson
mobility-report-nov-2015.pdf

make use of Emergent Configurations (ECs), which consist of a set of things that connect and cooperate temporarily through their functionalities, applications, and services, to achieve a user goal [1, 2].

In this paper, we present (i) IoT-FED: an approach for **F**orming and enacting **E**mergent configurations through **D**omain objects in the **IoT**. It exploits heterogeneous and independently developed things, IoT services, and applications modeled as Domain Objects (DOs), a service-based formalism [3]. Specifically, we present a process and an architecture realizing IoT-FED by leveraging an IoT platform, and a developer guideline. Additionally, we present (ii) a prototype we developed for validation purposes. The prototype is used to run experiments on two scenarios: one including (real/hardware) things is realized in our IoT lab[2] and the other where the hardware is simulated.

ECs are goal driven IoT systems, i.e., a user goal is the main driver to form and enact an EC. In this paper we assume that: *(A1)* ECs are formed and enacted within well-defined spatial boundaries (e.g., a room, a building). Therefore, we envision the number of things that might potentially form an EC to be in the scale of hundreds. This remarkably mitigates the well-known scalability issue in the IoT [4]. *(A2)* ECs achieve non-time critical user goals, i.e., ECs are formed and enacted within the time scale of seconds. *(A3)* ECs can be successfully formed and enacted, i.e., some needed things and services to achieve the goal are available and working until it is reached. Enabling the automated adaptation of ECs is out of the scope of this paper.

The Adjust Light Scenario. A concrete example of IoT systems we deal with is the Adjust Light (AL) Scenario that we will use throughout this paper.

(*AL-office*) Lara enters an office which is equipped with several IoT things including light sensors, and connected curtains and lights. The things, with their functionalities, are configured to be controllable by people in the office. Preparing for a meeting, she requests as goal to increase the light level using an application running on her smartphone. An EC formed by her smartphone, a light sensor, connected curtain and light is enacted to achieve her goal. The light sensor measures the light level in the room, the light is turned on and the curtain is partially opened. We realized this scenario, including both its hardware and software, in our IoT lab.

(*AL-hotel*) Lara enters a hotel room which is equipped with three light sensors, connected curtain and two connected lamps. The things, with their functionalities, are configured to be controllable by guests living in the room. Using the same application running on her smartphone, Lara expresses her goal to decrease light level in the room. Given that the curtain is closed, an EC formed by her smartphone, one of the light sensors and the two lamps is enacted to reach her goal. The light sensor measures the light level in the room, one of the two lamps is turned off and the light intensity level of the second lamp is reduced.

To realize the AL-scenario we leverage the Amazon AWS-IoT platform[3], which provides RESTful and publish-subscribe based APIs.

The remainder of the paper is organized as follows. Section 2 describes background notions of DOs. Section 3 illustrates IoT-FED in terms of its process, architecture, and a guideline. Section 4 presents the AL-office Scenario realized including hardware things, and the (software) prototype at work. Section 5 illustrates the validation of IoT-FED. In

[2] http://iotap.mah.se/lab/ [3] https://aws.amazon.com/iot/

Section 6 lessons learned are discussed. Section 7 surveys related works and Section 8 concludes the paper and draws future research directions.

2 Background on the Domain Object Model

The Domain Object model is the building block of a design for adaptation approach [3]. It allows to define independent and heterogeneous things and services in a *uniform way*, allowing developers to work at an abstract level without need to deal with the heterogeneity of things and protocols. In this section, we report the main features of the DO model that support the ECs formation and enactment, while other details can be found in [3, 5].

The intuition behind the DO model is to separate *what* a system is designed to do (e.g., adjust the light level) from *how* to do it (e.g., by combining available light sensor S and actuators A_1 and A_2). The how can vary in different execution contexts since can be provided by disparate things dynamically available. The AL scenario highlights a high degree of *heterogeneity* in terms of: types of things (e.g., sensors, actuators, smartphones) and their protocols (e.g., communication, data, infrastructure).

Things can be wrapped as DOs, e.g., by applications developers. This is done *una tantum*. After its wrapping, a thing is seamlessly part of the framework and exploited for the dynamic formation and enactment of ECs. In Figure 1, we provide a partial overview of the DOs model of the AL Scenario. Each DO implements a specific *domain property* which models a thing capability. For instance, the `light sensing` property models the light sensing capability of a light sensor. Each DO has a *core process* which defines its behavior. In addition, it exposes one or more fragments (e.g.,`checkActuatorsStatus` and `handleActuators`) which describe the functionalities it provides. Both core process and fragments are modeled as processes, by means of the *Adaptive Pervasive Flows Language* (APFL) [5]. Furthermore, each DO has *domain knowledge*, which represents its view on the environment in which it is. A DO's domain knowledge consists of: (i) internal knowledge that is the domain concept it implements e.g., the `Actuators management` in the `Actuators Manager`; (ii) external knowledge that is domain concepts it requires

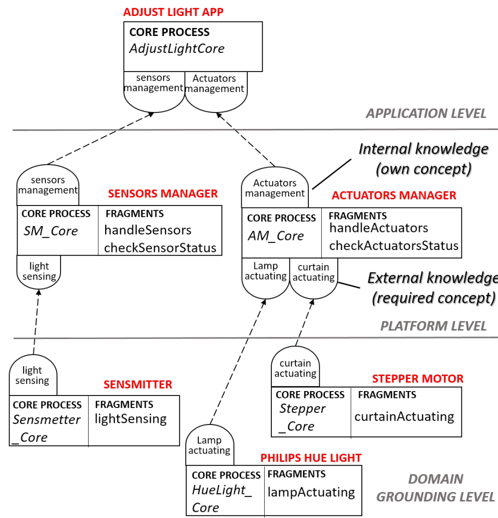

Fig. 1: Adjust Light Scenario: a partial overview of the DOs model of the system.

for its execution e.g., the `Lamp actuating` and `Curtain actuating` provided by other DOs.

Domain concepts are modeled as State Transition Systems (STS) [5]. The execution of fragments and core process of a DO updates STS state in its knowledge. *Domain-specific* concepts (e.g., light sensing) are usually implemented by DOs wrapping real things (e.g., sensors, actuators). *General-purpose* concepts (e.g., adjust light level), instead, might be realized by defining *value-added DOs* (e.g., a DO modeling the AL application of our scenario), which exploit the functionalities provided by the domain-specific ones.

Unlike traditional services, APFL allows the partial specification of the expected behavior of a DO by defining *abstract activities*: activities which the DO requires but it does not implement by itself. They are defined only in terms of a *goal* (e.g., sense the light). At runtime, the execution of abstract activities is performed through a *specialization* process, also called *refinement* mechanism. It allows abstract activities to be *refined* according to the fragments offered by other DOs, whose execution allows the abstract activity's goal to be reached. The so called *Higher Order Abstract Activities* (HOAA) enable a higher level of abstraction. They are used to dynamically generate abstract activities deriving from a performed reasoning activity, as explained in Section 4.2.

It is important to notice that fragments can be partially specified, too. Thus, they can also contain abstract activities. This enables a *chain of refinements* that we will detail in Section 4.2. The abstract activity refinement mechanism we deal with in this paper is performed through the application of advanced techniques for the dynamic and incremental service composition based on AI planning [5]. The AI planner takes as input the abstract activity's goal, the available fragments, the domain properties defined in APFL, and it returns a fragments composition process whose execution guarantees to reach the goal. A dynamic network of DOs is established at runtime to achieve the user goal. For instance, the dashed arrows in Fig. 1 stand for *potential* runtime relations that DOs might establish through the exchange of fragments (see Section 4).

3 The IoT-FED Approach

The IoT-FED approach includes (a) a process, (b) an architecture, and (c) a guideline. The IoT-FED is a refinement of the abstract architectural approach presented in [1] for enabling the automated formation and enactment of ECs in the IoT. Another possible refinement of [1] is presented in [6] where the focus is different compared to this paper, i.e., it is on the overall approach including adaptation, there is no use of DOs, and it exploits a different software prototype and scenario for experiments.

3.1 The IoT-FED Process

Figure 2 shows the IoT-FED process to enable the automated formation and enactment of ECs exploiting DO technologies. Labels A to H on process activities map them to one or more architectural components handling them in Fig. 3. The process starts by specifying the user *goal type* and the *goal spatial boundaries*. As already mentioned, the user expresses her/his goal via an application running on one of the available smart

device called the user agent. The goal type corresponds to the type of functionality provided by the application running on the user agent (e.g., adjust light level). The goal spatial boundaries correspond to the location where the EC must be formed and enacted (e.g., Lara's office). We consider the goal spatial boundaries to be the same boundaries of the user agent. Given these inputs, the core process of the DO corresponding to the specified goal type (e.g., `adjust light app` DO in Fig. 1) is loaded, and its simulation starts. This simulation is needed to check that an EC can be formed to achieve the user goal in the given spatial boundaries. The goal is achievable if all abstract activities in the core process can be refined successfully. All the core process activities are traversed, without being executed. When an abstract activity is found, a refinement process is performed to generate a plan (i.e., composition of fragments) which refines it - if any.

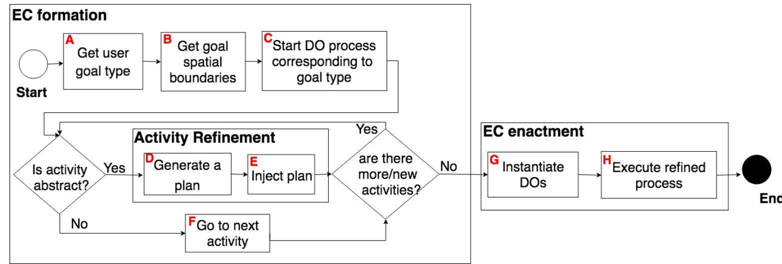

Fig. 2: The EC formation and enactment process

The generated plan, which might itself contain abstract activities, is then injected in place of the abstract activity it refines. If all found abstract activities are refined successfully, an EC exists and is formed. The formed EC is made up by the set of things whose corresponding DOs have been involved in the refinement process, through the selection of their fragments. The formed EC can then be enacted by instantiating the DOs forming it. Finally, the initial core process where all the generated plans are injected, can be executed. In this work, the fragments selection is purely functional. We plan to extend the process to support some situational context and dependencies in selecting fragments and consequently EC constituents.

3.2 An Architecture realizing IoT-FED

Figure 3 shows a possible refinement of the abstract architecture presented in [1]. The architecture realizes the IoT-FED process of Fig. 2, exploits existing components of the DOs technologies, and presents newly developed components. In the following, we describe the components and their responsibilities.

Goal Manager. It is responsible for parsing the user goal and starting the EC formation process. It has two sub-components: 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; the *Planner* responsible for the refinement of abstract activities in the loaded process.

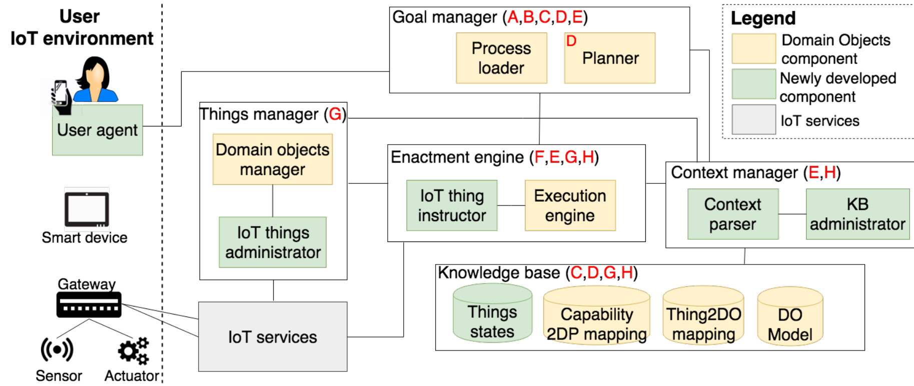

Fig. 3: The IoT-FED architecture

Things Manager. It is responsible for managing available IoT things and DOs. It includes: the *IoT things administrator* responsible for answering queries about available IoT things, their capabilities and locations; the *Domain objects manager* responsible for instantiating needed DOs and handling co-relations among them.
Enactment Engine. It is mainly responsible for enacting the EC and includes two subcomponents: the *Execution engine* is responsible for both (i) simulating and forming the EC - labels E and F in Fig. 2, and (ii) enacting it -labels G and H in Fig. 2. During the EC formation, it handles the injection of the plans received by the Planner in place of the abstract activities they refine, and during the EC enactment it executes the final refined process that achieves the user goal. The *IoT thing instructor* is responsible for passing the Execution engine instructions to IoT things (e.g., get sensor readings).
Context Manager. It is responsible for maintaining the system knowledge. It includes *KB administrator* responsible for retrieving data from the KB and the *Context parser* responsible for parsing received context from the Execution engine (e.g., new things states) and passing it to the KB administrator which updates the KB.
Knowledge base. It holds the internal system knowledge and includes four repositories: the *Things states* stores knowledge about things operational states (e.g., if lights are turned on or off); the *DO Model* that stores all the designed DOs; the Thing2DO mapping stores associations between things and the DOs representing them (e.g., the `Sensmitter` DO is a *Light Sensor* thing); the *Capability2DP mapping* that stores information about capabilities and domain properties relations. Different domain properties can relate to the same capability. For instance, the *actuate light level* capability can be mapped on both the `Curtain Actuating` and `Lamp Actuating` domain properties.
IoT services. The IoT-FED approach relies on a cloud platform to provide IoT services which enable the management and interaction with things. We leverage the Amazon AWS-IoT platform.

In general, realized components can be deployed to the Cloud (see Section 3.3), standalone servers or to hybrid infrastructures. A potential infrastructure shall posses capabilities of consuming available IoT services. The deployment model and aspects about security, connectivity, interoperability, and so on are out of the scope of this paper.

3.3 A Guideline about IoT-FED

To make IoT things and services available to be used through IoT-FED, a *developer* needs to do two main operations, here mapped on the development of the AL prototype.

1. *Register things in the AWS-IoT platform*: using the notion of templates in the AWS-IoT platform, developers can register different types of IoT things. For each thing, three searchable attributes should be specified: (thing) location, capabilities and REST endpoint automatically generated by the platform. When a thing is set up for the first time, a developer needs to store the thing initial state in the Things state repository. Finally, by exploiting the AWS-IoT services, the Context manager can dynamically answer queries about which things are available in a specific location, which of them have a specific capability and how to communicate with them.
2. *Model things, services and applications as DOs*: we illustrate this task by describing the modeled AL prototype in terms of DOs (see Fig. 1). The operational environment of the target system is defined through domain properties representing capabilities (e.g., light sensing, lamp actuating) that have to be registered in the Capability2DP mapping repository. Then, DOs implementing the specified domain properties have to be designed. The REST endpoints generated by the platform are invoked in the DOs processes activities. We can distinguish three levels in the DO model. At *domain grounding level* there are all the DOs of things offering functionalities and possibly requiring the interaction with third-party systems and things. DOs wrapping things have to be designed for each thing type and brand. For instance, the DOs of the things that can be part of a smart room such as the `Sensmitter` light sensor -with its fragments, e.g., `senseLight`. At *platform level* we have value-added DOs offering value-added services, such as the `Actuators Manager` in Fig. 1. These are defined through inter-dependencies among DOs implementing domain-specific functionalities, e.g., the handling of actuator devices. At *application level* we have the `AdjustLightApp` DO modeling as value-added DOs the user application used for adjusting the light level in different locations - while the user moves around. Each DO model has to be stored in the DO Model repository, and for those wrapping things, the mapping thing-DO is registered in the Thing2DO mapping repository.

4 The AL-office Scenario and AL Prototype Running on IoT-FED

This section presents: the realized AL-office Scenario including the (real/hardware) things in our IoT lab exploited in the developed (software) prototype, and the DOs dynamic mechanism leveraged for forming and enacting ECs.

4.1 The Realized AL-office Scenario

The IoT things we used to develop the AL-office Scenario are: three climate Sensmitters[4] including light sensors, a Philips Hue light[5] which has three bulbs connected by a

[4] https://www.senssolutions.se/ [5] https://www2.meethue.com/en-us

bridge, and motorized blinds we built consisting of a stepper motor[6] integrated with off the shelf blinds[7]. As said, to enable the management and interaction with the things, we leveraged the Amazon AWS-IoT platform and we deployed the architectural software components to an Amazon Elastic Beanstalk[8] instance running a Glassfish web server. Moreover, we use *gateways* as intermediaries between the AWS-IoT platform and the used things that do not have the processing and storage capabilities needed to connect directly to the platform (see Fig. 3). Particularly, the Sensmitters transmit data through a smartphone via Bluetooth technology. The hue light are connected through an Arduino[9] board, which also invokes the Philips Hue API services to adjust the bulbs light levels. Likewise, the motorized blinds connect through another Arduino board, which also sends the low level commands to rotate the stepper motor, thus opening or (partially) closing the blinds. The communications among the Enactment engine, Things manager and the IoT platform are MQTT[10] based. To develop the AL prototype, we followed the guideline about IoT-FED, as anticipated in Section 3.3. In the following section we provide an example of the AL prototype execution.

4.2 Running the AL Prototype on IoT-FED

When a user expresses a goal (at runtime) via a user agent, this triggers the execution of the IoT-FED process (Fig. 2). The execution starts from the `AdjustLightApp` DO, corresponding to the AL application running on the user's smartphone. Here, we describe the key mechanism contributing to the formation and enactment of ECs, i.e., the dynamic refinement of DOs' abstract activities through the dynamic discovery, injection, and execution of fragment-based plans.

Figure 4 depicts the situation in which Lara enters her office and she wants to adjust the light level in the room. We suppose that this can be done by setting a slider bar (e.g., from 0 to 10 level) on the AL application interface. Figure 4 (a) describes the *EC formation* process of Fig. 2 performed by simulating the execution of the AL core process. All its activities are traversed, looking for abstract activities and checking that they can all be successfully refined through some composition of fragments. If this is possible, at least one EC exists and the generated refinement is injected in the original AL core process. Figure 4 (b), instead, describes the *EC enactment* process of Fig. 2 where the generated refined AL core process can be executed. Note that, for the sake of description, in the following we mix details of ECs formation and enactment. However, remind that the EC enactment starts only if and after an EC is successfully formed. The EC formation starts from the core process of the AL application[11]. When its execution starts, information about the room where the user is located are retrieved, in order to get the goal spatial boundaries. Then, among others, a sequence of two abstract activities (represented with dotted lines and labeled with a goal defined on top of domain properties) need to be refined. They refer to the sensing and the actuating of the light level in the room, respectively. Then, the refinement mechanism is triggered. In the following we list the refinement steps performed while forming the EC.

---

[6] https://components101.com/motors/28byj-48-stepper-motor

[7] https://jysk.se/gardiner/persienner/aluminium/alu-persienn-60x80cm-vit

[8] https://aws.amazon.com/elasticbeanstalk [9] https://www.arduino.cc/ [10] http://mqtt.org/

[11] For presentation purposes, in Fig. 4 we omit some details.

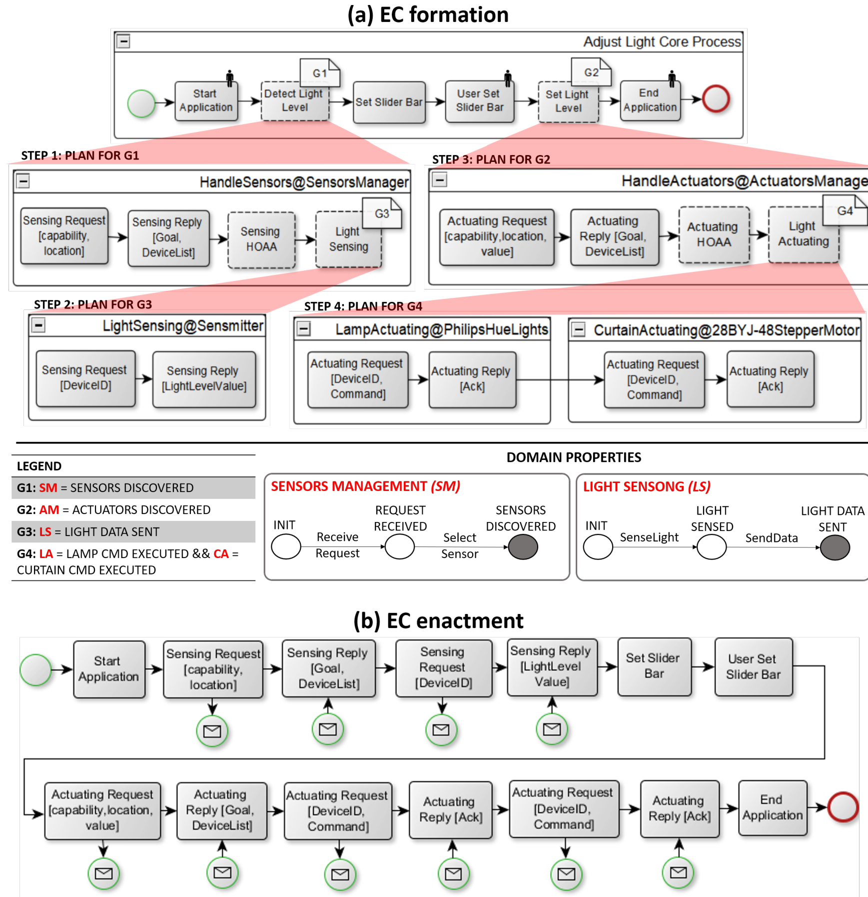

Fig. 4: AL Scenario: an example of the *EC formation* (a) and the *EC enactment* (b).

Step 1 consists in finding a plan for the goal `G1: SM = Sensor Discovered`. To this aim, the fragment `HandleSensors` from the `SensorsManager` is selected for the refinement and injected in the AL process, in place of the `Detect Light Level` abstract activity. During the EC enactment phase, the execution of this fragment allows to send a `Sensing Request`, by specifying the required capability (e.g., sense *light* level), and receive information about the list of available things that might be exploited for the request. The `Sensing Higher Order Abstract Activity` dynamically generates the `Light Sensing` abstract activity, after the application understands that the required capability deals with the sensing of the light, given that the `SensorsManager` handles different kinds of sensors.

Step 2 consists in finding a plan for the dynamically generated goal `G3: LS = Light Data Sent`, which has been defined over the `Light Sensing` domain property. It is interesting to notice that the `Sensors Manager` is not able to provide itself the required fragment for the light sensing, because the sensing process depends on the

thing used to do it. Only during the execution, the system can discover and select the proper things it needs, in the specific context in which it is running. In the Lara's office room, the `Light Sensing` fragment is given by the `Sensmitter` sensor.

Step 3 and **step 4** are performed by following the same procedure of steps 1 and 2, with the only difference that the system needs now to discover actuators to set Lara's desired light level. Specifically, a composition made by fragments offered by the `Philips Hue lights` and the `Stepper Motor` actuators installed in the office is used to reach the user goal.
The final AL application process executed to enact the AL EC is the one in Fig. 4 (b).

5 Validation

To evaluate the *feasibility* of IoT-FED, we conducted two experiments aiming at answering: *can IoT-FED be used to handle the dynamic formation and enactment of ECs at runtime?* Table 1 presents the specifications of two rooms corresponding to the *AL-office* and the *AL-hotel* scenarios (see Section 1). For each room, we defined the installed things, i.e., light sensors and actuators. Then, we wrapped the things as DOs and made them usable by the IoT-FED following the guidelines presented in Section 3.3. Note that the feasibility of the domain object-based approach is analyzed in [7]. The time for a developer to wrap a service as DO is paid only una tantum; it varies depending on her/his knowledge of DOs and expertise level and in the worst case is few hours.

Table 1: Office and hotel room's stationary things.

Device Type	Office Room	Hotel Room
Light Sensors	Sensmitter, LightMeter, WiStar, SensLight	Sensmitter, DayLight, WiStar
Lamp Actuators	PhilipsHueLight, SmartLamp	PhilipsHueLight, IkeaLamp
Curtain Actuators	HS-422-ServoMotor, StepperMotor	HS-422-ServoMotor

Our experiments include 11 different things of various brands. Besides the real world things (e.g., Philips Hue Light) we used to implement the AL-office scenario, we modeled additional things to simulate more complex still realistic settings. The office and hotel rooms are equipped with 8 and 6 different stationary things, respectively, in addition to Lara's smartphone. We performed sequential executions of $< 100 >$ runs of the AL application per each room to: (i) evaluate the IoT-FED ability to dynamically generate different ECs based on available things; (ii) get insights on the IoT-FED performances. We evaluated our approach using a dual-core CPU running at 2.7GHz, with 8Gb memory.

Figure 5 shows the different dynamically generated ECs and their occurrences over the 100 runs of the AL application in the hotel room. Based on randomly requested light levels at each run, 14 different ECs were formed and enacted. For instance, the EC number 1 is made as follows: $EC1 = \langle smartphone, DayLight, HS-422-ServoMotor, IkeaLamp \rangle$ and it occurs 9 times over the 100 runs. Compared with the hotel room, the office room has one more light sensor and one more curtain actuator. Figure 6 illustrates that with only two more devices in the domain space, 32 different ECs can be formed[12]. For instance, the EC number 2 is made as follows: $EC2 =$

[12] For readability purposes, in Fig. 6, we show only a subset of the 32 ECs.

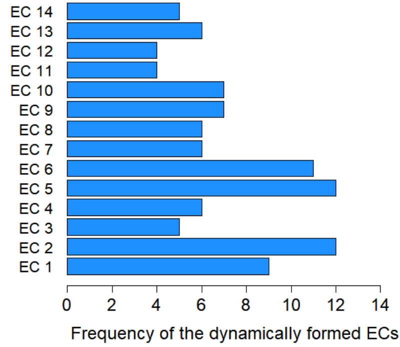

Fig. 5: ECs in the hotel room.

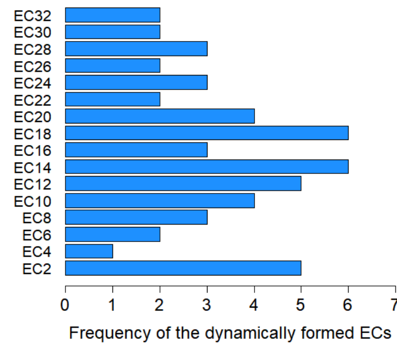

Fig. 6: ECs in the office room.

$\langle smartphone, LightMeter, HS-422-ServoMotor, PhilipsHueLight\rangle$ and it occurs 5 times over the 100 runs. For each EC occurrence, we also measured the time required to form and enact it. Fig. 7 shows the average time for forming and enacting ECs over the 100 runs of the AL scenario for both the hotel and office rooms, i.e., 2.39 secs and 2.44 secs respectively.

One of the advantages of managing IoT domains w.r.t. Internet of Services (IoS) domains is that we deal with simpler DOs processes and fragments (i.e., made by few activities and transitions). This positively impacts on the time required for the dynamic composition of fragments and keeps it in the order of milliseconds. In summary, we can draw positive conclusions about the feasibility of the IoT-FED approach in forming and enacting ECs in dynamic environments. According to the assumption *(A1)* in Sec. 1, we considered a realistic set of things in these experiments, based on the spatial boundaries where ECs are enacted. While for future work we plan to apply the approach to additional real-world scenarios involving more things.

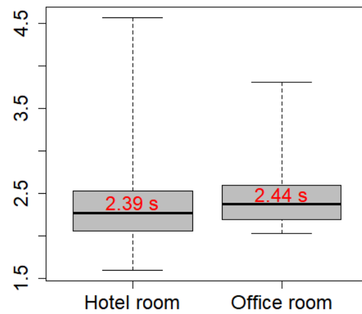

Fig. 7: ECs execution time per room.

6 Lessons Learned

DOs formalism. IoT things are heterogeneous and operate using different standards. This makes it evident that there is a need for other layers which can handle complex processes such as enabling ECs formation and enactment. Wrapping things as DOs has several *advantages*. For instance, as DOs, things are represented in a uniform way, allowing developers to work at an abstract level where they do not always need to deal with the heterogeneity of things and protocols. Nevertheless, the DOs formalism can be further *extended* to better meet IoT domain's features and requirements. Particularly, when refining an abstract activity, the fragments discovery and selection is currently functional. Available fragments whose execution allows the abstract activity's

goal to be reached are selected and composed. However, specially in IoT domains, a non-functional selection process would be more appropriate (e.g., to select the device which has a good battery level instead of a random one).

Moreover, the Execution Engine currently operates in a centralized manner. It should evolve to better deal with the execution of distributed systems. Since IoT things are wrapped as DOs manually, the IoT-FED approach supports expressing a set of goals whose types are specified at design time. We plan to support the automated wrapping of things as DOs, thus enabling the definition of new goal types at runtime. Currently, runtime failures require re-executing the EC formation and enactment process to form new ECs which maintain the achievement of user goals. We recall that enabling automated adaptation of ECs is out of the scope of this paper.

Amazon IoT platform and deployment services. On the one hand, the AWS-IoT platform contributes several *advantages* to the IoT-FED approach. For instance, it provides APIs which support managing and interacting registered IoT things. The platform also supports routing a huge number of messages at runtime. On the other hand, the platform imposes some *limitations* and restrictions. For instance, it puts sophisticated security requirements for registering IoT things. Although the AWS Elastic Beanstalk facilitates the deployment process, it may require developers to perform security related configurations when applications are (re-)deployed to web servers.

7 Related Work

In the context of architectures, the IoT-FED architecture is compliant with the IoT reference architecture proposed in [8]. The SOCRADES Integration Architecture (SIA) is a SOA architecture designed to couple the IoT with enterprise services [9]. In SIA, processes are modeled at design time by using BPEL. The authors also extended the BPEL language to enable the dynamic assignment of services at the execution phase. In [10], the authors propose a service-based architectural approach to enable efficient and adaptive composition of services. Composite services are modeled and specified at design time, thus limiting systems flexibility.

The usage of business processes with their related technologies in the IoT context [11] is a novel research field that opened interesting research challenges [12]: from extensions of standard workflow languages (i.e., WS-BPEL, BPMN 2.0) proposed to support suitable communication paradigms for the IoT [13, 14], to workflow management systems (WfMS) for industrial IoT [15, 16] to execute and monitor IoT-based processes. WfMS suitable in dynamic contexts have also been proposed, to adapt processes in case of failures by replacing the respective resources (things or services) or workflow tasks [17, 18]. The use of the APFL language enables the IoT-FED approach to dynamically refine abstract processes with concrete ones provided by available things, thus making it suitable in dynamic contexts, too.

The growing number of online resources, and services led to the rise of methodologies and tools to create applications by combining them, referred to as *mashups*. Most mashups approaches focus on the composition of web-based interfaces and functionalities. In [19], the authors focus on the importance of *context-awareness* and *adaptivity* of service mashups in dynamic environments, otherwise, they tend to be misaligned

with their execution environments. In [20], the authors objective is that of overcoming the static nature of IoT applications that, although highly responsive, are usually based on pre-compiled mashups, being thus inflexible. The work is based on a decentralized goal-driven composition of pre-compiled service mashups and, similarly to IoT-FED, it relies on the abstraction of the referring environment allowing to abstract service composition requirements. In last years, to facilitate the development of IoT application, different mashups editors have been proposed, such as e.g., [21, 22], providing developers a visual support abstracting both things and services they can compose together. Differently from [22] our approach includes, but is not limited to, the use of REST services. Moreover, it allows applications to define dynamic behaviors, due to the use of abstract activities refined through services composition when the context is known or discovered. The trade-off of the majority of service mashups approaches in the literature is that applications rely on static mashups that cannot deal with open environments. Service-oriented architectures for planning, execution and adaptation of cyber-physical systems (CPS) have been proposed in [23, 24]. The approach proposed in [23] is based on a clean separation between domain modeling, planning, execution, monitoring and actuation services enabling the realization of large scale CPSs. In [24], the authors propose a MAPE-K autonomic computing framework to manage adaptivity in service-based CPSs.

Several approaches discuss coalitions of coordinating components, known as *choreography*. For instance, in [25], a formal framework dealing with self-adaptation in the context of choreographies is proposed. A group of interacting components has also be seen as an *ensemble*. Languages to define ensembles have been introduced (i.e., [26, 27]) to specify what groups of components should be present in the system together with mechanisms to select at runtime the ones satisfying specific constraints (i.e., predicates). In the IoT-FED approach, the system configuration is less-constrained by design and it leaves components to freely interact to achieve a user goal in a specific context.

8 Conclusion and Future Work

In this paper, we presented IoT-FED, an architectural approach enabling the automated and runtime formation and enactment of ECs in dynamic environments. The approach exploits the DO model and its technologies in the context of IoT environments. We have presented both the process and the architecture of the IoT-FED approach, with a guideline about how to use it. Additionally, we presented the developed prototype used for running initial validation experiments, which showed positive and promising results.

In the future, we plan work in a number of directions including: extending IoT-FED to deal with the adaptation of ECs at runtime (leveraging the adaptation mechanisms of DOs); identifying and realizing additional realistic cases that deals with multiple and possibly competing ECs and using them for more extensive validation; extending the approach to deal with the selection and composition of things based on some non functional aspects.

Acknowledgment

This work was partially funded by the project SmartConstruction[13] (EIT Digital activity #18014) and by the Knowledge Foundation through the Internet of Things and People research profile (Malmö University, Sweden).

References

[1] Alkhabbas, F. and Spalazzese, R. and Davidsson, P.: Architecting Emergent Configurations in the Internet of Things. In: IEEE International Conference on Software Architecture, pp. 221–224. IEEE, 2017.
[2] Ciccozzi, F. and Spalazzese, R.: MDE4IoT: Supporting the Internet of Things with Model-Driven Engineering. In: 10th Intern. Symp. on Intelligent Distributed Computing, pp. 67–76. Springer, 2017.
[3] Bucchiarone, A. and Sanctis, M. De and Marconi, A. and Pistore, M. and Traverso, P. : Design for Adaptation of Distributed Service-Based Systems. In:International Conference on Service-Oriented Computing, pp.383–393. Springer, 2015.
[4] Atzori, L. and Iera, A. and Morabito, G.: The internet of things: A survey. In: Computer Networks, pp. 2787–2805. Elsevier, 2010.
[5] Bucchiarone, A. and Sanctis, M. De and Marconi, A. and Pistore, M. and Traverso, P.: Incremental Composition for Adaptive by-Design Service Based Systems. In: IEEE International Conference on Web Services, ICWS. IEEE, 2017.
[6] Alkhabbas, F. and Spalazzese, R. and Davidsson, P.: ECo-IoT: an Architectural Approach for Realizing Emergent Configurations in the Internet of Things. In: European Conference on Software Architecture (ECSA), to appear. Springer, 2018.
[7] Bucchiarone, A. and Sanctis, M. De and Marconi, A.: ATLAS: A World-Wide Travel Assistant Exploiting Service-Based Adaptive Technologies. In: International Conference on Service-Oriented Computing, pp.561–570. Springer, 2017.
[8] Bauer, M. et al.: IoT Reference Architecture. In: Enabling Things to Talk: Designing IoT solutions with the IoT Architectural Reference Model, pp.163–211. Springer, 2013.
[9] Spiess, P. and Karnouskos, S. and Guinard, D. and Savio, D. and Baecker, O. and De Souza, L.M.S. and Trifa, V.: SOA-based Integration of the Internet of Things in Enterprise Services. In: IEEE International Conference on Web Services, pp. 968–975. IEEE, 2009.
[10] Dar, K. and Taherkordi, A. and Rouvoy, R. and Eliassen, F.: Adaptable Service Composition for Very-Large-Scale Internet of Things Systems. In: 8th Middleware Doctoral Symposium, pp. 2. ACM, 2011.
[11] Chang, C. and Srirama, S. N. and Buyya, R.: Mobile Cloud Business Process Management System for the Internet of Things: A Survey. In: ACM Comput. Surv., vol.49 pp.70:1–70:42. 2017.
[12] Janiesch, C. et al.: The Internet-of-Things Meets Business Process Management: Mutual Benefit and Challenges. In: arXiv preprint arXiv:1709.03628, vol. abs/1709.03628. 2017.

[13] https://enoba.eu/projects/smartconstruction

[13] Domingos, D. and Martins, F. and Cândido, C. and Martinho, R.: Internet of Things Aware WS-BPEL Business Processes Context Variables and Expected Exceptions. In: J. UCS, vol. 20 pp.1109–1129. 2014.
[14] Tranquillini, S. et al.: Process-Based Design and Integration of Wireless Sensor Network Applications. In: International Conference Business Process Management, pp. 134–149. Springer, 2012.
[15] Mass, J. and Chang, C. and Srirama, S.N.: WiseWare: A Device-to-Device-Based Business Process Management System for Industrial Internet of Things. In: IEEE International Conference on Internet of Things, pp.269–275. IEEE, 2016.
[16] Seiger, R. and Huber, S. and Schlegel, T. : Toward an execution system for self-healing workflows in cyber-physical systems. In: Software and System Modeling, vol. 17, pp. 551–572. 2018.
[17] Seiger, R. and Huber, S. and Heisig, P.: PROtEUS++: A Self-managed IoT Workflow Engine with Dynamic Service Discovery. In: Central European Workshop on Services and their Composition, pp. 90–92. 2017.
[18] Wieland, M. and Schwarz, H. and Breitenbücher, U. and Leymann, F.: Towards situation-aware adaptive workflows: SitOPT - A general purpose situation-aware workflow management system. In: IEEE International Conference on Pervasive Computing and Communication, pp.32–37. IEEE, 2015.
[19] Dorn, C. and Schall, D. and Dustdar, S.: Context-aware adaptive service mashups. In: IEEE Asia-Pacific Services Computing Conference, pp. 301–306. IEEE, 2009.
[20] Ciortea, A. and Boissier, O. and Zimmermann, A. and Florea, A. M.: Responsive Decentralized Composition of Service Mashups for the Internet of Things. In: 6th ACM International Conference on the Internet of Things, pp. 53–61. ACM, 2016.
[21] Giang, N. Ky and Blackstock, M. and Lea R. and Leung, V. C. M.: Developing IoT applications in the Fog: A Distributed Dataflow approach. In: International Conference on the Internet of Things, pp. 155–162. IEEE, 2015.
[22] Mayer, S. and Verborgh, R. and Kovatsch, M. and Mattern, F.: Smart Configuration of Smart Environments. In: IEEE Transactions on Automation Science and Engineering, pp. 1247–1255. IEEE, 2016.
[23] Feljan,A. V. and Mohalik, S. K. and Jayaraman, M. B. and Badrinath, R. : SOA-PE : A Service-oriented Architecture for Planning and Execution in Cyber-physical Systems. In: International Conference on Smart Sensors and Systems, pp. 1–6. IEEE, 2015.
[24] Mohalik, S. K. and Narendra, N. C. and Badrinath, R. and Le, D.: Adaptive Service-Oriented Architectures for Cyber Physical Systems. In: IEEE Symposium on Service-Oriented System Engineering, pp. 57–62. IEEE, 2017.
[25] Coppo, M. and Dezani-Ciancaglini, M. and Venneri, B.: Self-Adaptive Monitors for Multiparty Sessions. In: IPDP, pp. 688–696. IEEE, 2014.
[26] Krijt, F. and Jirácek, Z. and Bures, T. and Hnetynka, P. and Gerostathopoulos, I.: Intelligent Ensembles - A Declarative Group Description Language and Java Framework. In: IEEE International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pp. 116–122. IEEE, 2017.
[27] De Nicola, R. and Loreti, M. and Pugliese, R. and Tiezzi, F.: A Formal Approach to Autonomic Systems Programming: The SCEL Language. In: TAAS. 2014.