Automated Deployment of Data Collection Policies over Heterogeneous Shared Sensing Infrastructures

Cyril Cecchinel, Sébastien Mosser, Philippe Collet

Automated Deployment of Data Collection Policies over Heterogeneous Shared Sensing Infrastructures

Cyril Cecchinel
Université Côte d'Azur
CNRS, I3S, France
Email: cecchine@i3s.unice.fr

Sébastien Mosser
Université Côte d'Azur
CNRS, I3S, France
Email: mosser@i3s.unice.fr

Philippe Collet
Université Côte d'Azur
CNRS, I3S, France
Email: collet@i3s.unice.fr

***Abstract*—Smart buildings and smart cities rely on interconnected sensor networks that collect data about their environment to support various applications. Developing and deploying the data collection architectures of these systems is a challenging problem. The specificities of the sensor platforms compel software engineers to work at a low level. This makes this activity tedious, producing code that badly exploits the network architecture, and hampering reuse of data collection policies. Moreover, several data collection programs cannot be guaranteed to be deployable on a shared infrastructure. In this paper, we present an automated approach that supports *(i)* the definition of data collection policies at a higher level of abstraction, *(ii)* the representation of the diverse platforms and the network topology, and *(iii)* the automatic composition and deployment of the policies on top of heterogeneous sensing infrastructures following different strategies. The approach is tooled and has been assessed on both realistic and simulated deployments.**

***Index Terms*—sensor network; automated deployment; infrastructure reuse;**

I. Introduction

Physical objects interconnected through the Internet form the *Internet of Things* (IoT) [11]. According to the Gartner group, by 2020, up to 26 billions of these things could be available, producing huge information flows from diverse and large-scale sensing infrastructures. Smart buildings and smart cities are typical contexts in which many new applications have to be developed by relying on these infrastructures to collect data about their environment [26], [1]. In this context, pieces of software that manage and consume such data are extremely specific and dedicated to very different and evolving purposes, *e.g.*, transportation regulations, road traffic analysis, autonomic buildings. These application domains require their stakeholders (*e.g.*, domain experts) to define requirements that will be implemented by software engineers. The main challenge here is the gap that naturally exists between what a software engineer knows about the sensing infrastructures and how the relevant data can be collected [16].

For now, a sensor network needs to be configured at the hardware level and programmed at a low level according to the expressed needs. This activity is extremely tedious for software engineers as they need to understand the diverse architectures of sensor networks and deal with low-level programming languages (*e.g.*, C, nesC [18]) to find a common ground. Moreover, sensor network platforms evolve quickly, and it is out of the scope of a given application domain to follow such evolution. Finally, as sensor networks are designed for specific domains, it is not immediate to reuse a given network and adapt it for another purpose. As a consequence, sharing already deployed infrastructure is tedious, and software evolution on top of it is almost impossible when combined with the heterogeneous nature of the devices.

Developing and deploying the data collection architectures of these systems is a challenging problem. The current state of practice and the specificities of the sensor platforms compel software engineers and sensor network experts to work jointly at a very low level of abstraction to perform these tasks. Programming abstractions hopefully exist, but are linked to specific sensor operating systems of dedicated platforms [9], [18]. However these abstractions do not remove the need for code to embed network specific concerns, which are heterogeneous by nature.

Toolchains for the IoT have also been proposed, but they either focus on the user-centric applications [21] or the management of the underlying platforms [19]. Consequently they are not capable of supporting the composition and deployment of several policies on a shared infrastructures.

On the contrary some other approaches really focus on modeling data collection policies. However some of them do not abstract enough the low-level elements to provide an appropriate support for software engineers interacting with domain experts [20]. Others abstract too much with activity diagrams and low-level specific code must be written with no support for reuse and no handling of the diversity of sensing infrastructures [10]. Finally, when the concerns of the different stakeholders are well separated, the expression of data collection policies is not well supported and reuse is impossible [28]. With the increasing number of scenarios to be implemented over various infrastructures, the development and deployment activities then become more tedious and time-consuming.

To tackle these problems, we present in this paper a tooled approach that supports *(i)* the definition of data collection policies at a higher level of abstraction (for software engineers), *(ii)* the representation of the diverse platforms and the network topology (for sensor network experts), and *(iii)* the automatic composition and deployment of the policies on top of heterogeneous sensing infrastructures, following different strategies.

To achieve this envisioned approach in an operational toolchain, we identified the four following requirements (R_i)

expected by the different stakeholders.

R_1. **A software engineer handles only business concerns**. For now, the sensor network needs to be configured at the hardware level. Thus, software engineers need to understand the underlying infrastructure and use low-level programming languages. It is the responsibility of sensor network experts to abstract the complexity of such infrastructures. Such an abstraction also enables a software engineer to reuse her policy across different sensing infrastructures as her code is no more coupled with a specific sensor network.

R_2. **A data collection policy must be automatically adapted to the different platforms specificities**. The large number of available platforms has led to a large software and hardware diversity making each deployment specific to a single infrastructure. The state of the art already provides some abstractions but they need to be supported by the different platforms and remain dependent on the underlying infrastructure. To assist software engineers in the exploitation of a sensing infrastructure, a data collection policy should be automatically adapted to the specificities of the platforms.

R_3. **A data collection policy must be automatically projected over a sensing infrastructure**. With regard to the separation of concerns principles [13], a data collection policy must not be coupled with a sensing infrastructure. Large scale sensing infrastructures make available a multitude of platforms distributed in complex network topologies. A software engineer should not have to deal with this complexity, every concept of the data collection policy should be directly assigned to a sub-set of platforms. Every concept would then, according to a deployment strategy defined by a sensor network expert, be projected and instantiated on a single platform.

R_4. **An infrastructure must be automatically shared between different applications**. Considering the important investment required by the installation of a sensor network[1], reusing the infrastructure for multiple applications is necessary. At the time of the deployment of a new policy, several other policies can be already executed on the sensing infrastructure. Because of the separation of the software and network points of view, a software engineer cannot know a priori the other policies she needs to compose. Thus, automated deployment mechanisms should also provide automated composition capabilities.

II. Walkthrough

To demonstrate the need of an automated approach, we introduce here a running example[2] inspired from a realistic deployment [6]. We consider a software engineer who is in charge of programming an existing sensing infrastructure to implement several policies gathered from domain experts. The example focuses on a first policy that aims at monitoring energy losses in an office, *i.e.*, receiving an alert each time both the door and the window of an office are opened while the air conditioning is powered on. This scenario will be deployed on a sensing infrastructure that is currently executed on an academic campus where offices are equipped with door and window opening sensors, and air conditioning sensors. To simplify the exploitation of sensing infrastructure by software engineer, we advocate that the definition and the deployment of a data collection policy should respect the principles of separation of concerns, *i.e.*, a software engineer only focuses on the application logic while a sensor network expert provide the network logic (cf. requirement R_1). In this section, we propose to discuss each step of what would be an appropriate automated toolchain for programming sensing infrastructures while meeting the other three requirements identified in the introduction.

A. Toolchain Overview

As inputs, the toolchain consumes a data collection policy (provided by the software engineer) and a description of the network (provided by the sensor network expert). As output, it provides ready-to-flash code for each platform involved in the realization of the data collection policy.

The data gathering on a sensor network is configured using data collection policies, *i.e.*, sets of operations performed on data to convert them into knowledge [12]. As discussed in the introduction, sensing infrastructures are configured at the hardware level using low-level programming languages. This activity is tedious for software engineers as they need to understand the architecture of the network and deal with low-level programming languages [9], [18]. Other proposed abstractions are also not meeting all identified requirements at the same time [21], [28]. Besides the paradigm of wireless sensor network virtualization seems promising to abstract the complexity of the architecture but the state of the art does not provide a software engineer-oriented language [16].

To meet R_1, we propose that the software engineer relies on a Domain-Specific Language (DSL) to express her data collection policies over a sensing infrastructure. The software engineer expresses *(i)* properties of the data collection policy (*e.g.*, name, data-type), *(ii)* business operations and *(iii)* data flows between business operations. A graphical representation of this data collection policy for a given office (office 443 in the example) is presented on the left of Fig. 1.

Traditional deployment of such data collection policies is performed by considering *(i)* the facilities of each platform, *(ii)* the network topology and *(iii)* a deployment strategy.

Description of platform facilities. Any platform in a sensor network is built around five main components [15]: *Controller*, *Memory*, *Sensors*, *Communication* and *Power supply*. A data collection policy must be implemented according to these components. Because of the large number of manufacturers available on the market, each of those components can differ from a platform to an other, introducing a multiple variability problem.

The *Software Product Line* (SPL) paradigm allows one to manage this variability [23], especially by describing the elements of a family of systems in terms of features [4], [3].

[1] *e.g.*, 15 M€ have been invested for the Smart Parking infrastructure in the city of Nice, France (http://enstotoday.com/parking-worth-paying-for/).

[2] More details can be found on a companion web page: https://github.com/ace-design/DEPOSIT/blob/master/publications/apsec16.md

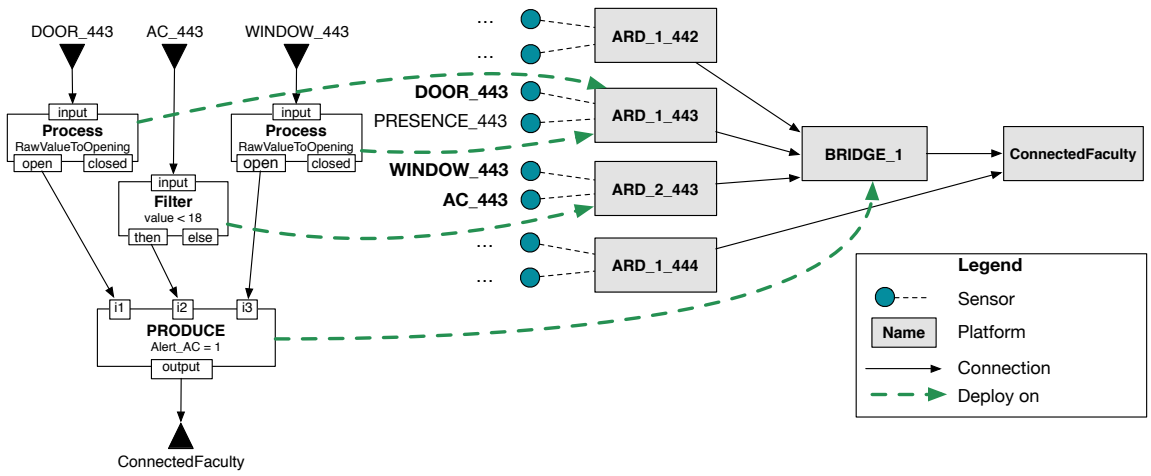

Fig. 1. Projection of a data collection policy on the CONNECTEDFACULITY topology

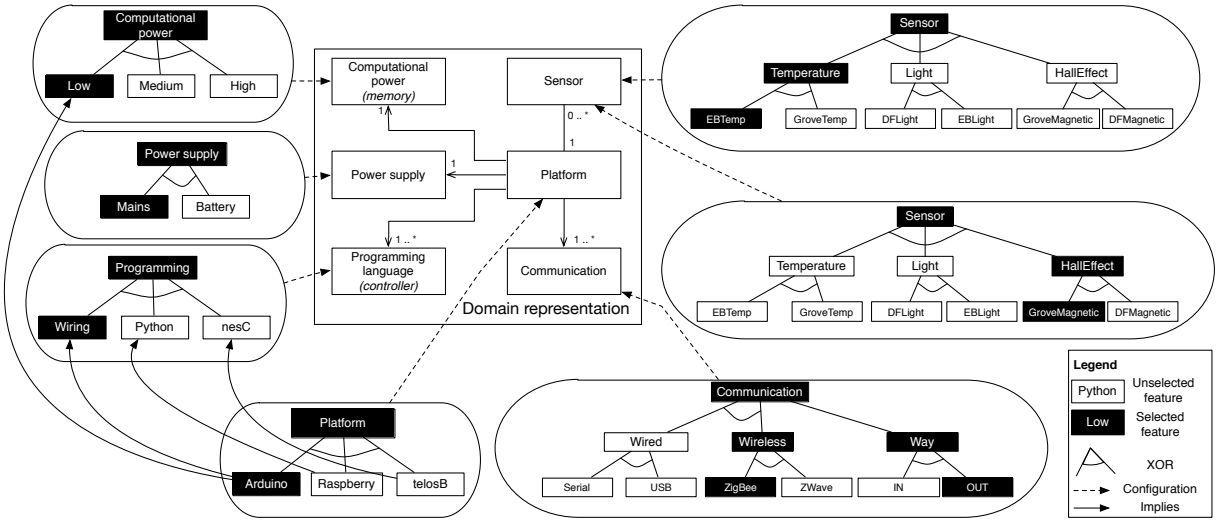

Fig. 2. Excerpt of a configuration describing a platform

Several works have notably been conducted in the definition of SPLs for other aspects of the Internet of Things [2], [22]. We thus capture the variability of the platforms through variability models, using the support of feature models [14] and associated automated analysis on these descriptions [5]. Figure 2 illustrates the individual configuration from a feature model of each component of a platform.

Network topology. The data collection policy must now be deployed over the sensing infrastructure. To automate this task, the toolchain relies on a model describing the network topology. The network topology is assimilated to a graph where vertices represent platforms and edges represent a connectivity between two platforms [27]. In our approach, we reuse the concept of directed graphs to model the sensing infrastructure.

Deployment strategy. A sensor network expert can tune the deployment strategy to optimize the data collection policy distribution over the underlying infrastructure. In the toolchain, we define two kinds of deployment strategies: *(i)* static strategies (S) and *(ii)* dynamic strategies (D). We define a static strategy as a strategy placing the concepts according to a static representation of the network (*e.g.*, *Closest to the sensors* strategy). Dynamic strategies deploy the data collection policy according to the current state of the sensing infrastructure (*e.g.*, *use in priority free platforms* strategy). While the toolchain provides the building blocks to encode different kinds of strategies, we have highlighted several deployment strategies corresponding to some relevant operating scenarios:

- *Closest to sensors (S).* Data collection policy's concepts are distributed as close as possible to sensors. This strategy allows one to maximize the use of computational resources offered by the sensing infrastructure.
- *Use in priority mains power supplied platforms (S).*

The energy-consumption in sensing infrastructure is a major concern for sensor network expert [30], [25]. This strategy maximizes the use of mains power supplied platforms in a network composed by both battery and mains powered elements.

- *Use in priority free platforms (D).* This strategy focuses on the deployment on platforms that have not been already involved in a previous deployment.

As an example, figure FIG. 1 illustrates the deployment of a data collection policy using the *closest to sensors* strategy.

B. Handling Multiple Applications

When reusing a sensing infrastructure, a software engineer may not have the knowledge of previously deployed applications as they might have been defined and deployed by others. A classical way to tackle this issue is to rely on isolation where nodes can execute at same time different applications, as it is achieved in the virtual machine isolation requirement for cloud computing.

Despite the need for infrastructures handling multiple applications simultaneously [16], nodes deployed in sensing infrastructures are most of the time resource-constrained devices and do not support multi-threading operations. To tame the complexity of sharing a sensing infrastructure (requirement R_4), the toolchain is able to compose new policies with previously deployed ones in a transparent manner for the software engineers. In our toolchain, a third-party service in charge of deploying policies on top of a given infrastructure, providing a composition mechanism (*cf.* SEC. III for more details).

III. TOOLCHAIN INTERNALS

A. Separation of Concerns

The toolchain leverages the separation of concerns principle by regrouping software engineers's concerns in a *data collection policy* and the sensor network experts's concerns in an *infrastructure model*.

a) Data collection policy: As stated in SEC. II, the DSL must only focus on business concerns. In order to meet R_1, we advocate that workflows are a suitable abstraction as they provide a convenient way to define *"a sequence of activities performed in a business that produces a result of observable value to an individual actor of the business"* [17]. A policy is then expressed by the software engineer by refining activities as her own concepts, and dependencies as a data flow between these concepts.

A data collection policy d relies on concepts $c \in C = concepts(d)$, clustered as sensors (*e.g.*, an electronic device transforming a physical quantity into a measurement, $s \in S = sensors(d)$), collectors (*e.g.*, remote Web service where processed data are sent for storage, $l \in L = collectors(d)$) and operations ($o \in O = operations(d)$). Operations refer to the domain expert's business concerns. A special kind of operation, *Process*, leverages the hierarchical workflow mechanism and allows software engineer to encapsulate a data collection policy into another, so to ease reuse. A data flow $f(c_a, c_b) \in F = flows(d)$ models the flows between a source concept c_a and a destination concept c_b. We then define a data collection policy as follows:

$$d = (name, concepts, flows) \in (Symbol \times C^\star \times F^\star)$$

b) Infrastructure model: The infrastructure model needs to describe the features available on each platform, the network topology, and the heuristics to be used to implement deployment strategies.

Platform features description model. We reuse the work of Urli *et al.* [29] associated with its graphical user interface[3] to capture complex configurations of platforms. The feature description model FD allows a sensor network expert to describe the specificities of a platform P and its connected sensors S (*e.g.*, FIG. 2) while providing a dynamic connection between them. This allows for a more flexible description, which also enables platforms to be changed more easily.

Network topology model. The network topology model I allows a sensor network expert to describe the relationships between platforms $P \subset F$. With this model is provided the $accessible$ operator that returns the subset of sensors S that can be reached from a platform, *i.e.*, $accessible : P \rightarrow S$. On FIG. 1, $accessible(ConnectedFaculty)$ returns all the sensors declared in the sensing infrastructure and the function $accessible(BRIDGE\_1)$ returns all the sensors excepting those connected to platform ARD_1_444.

Deployment strategy. A deployment strategy implements a $place$ operator that has been defined by the sensor network expert or reused from the provided deployment strategy presented in SEC. II. This operator maps a concept $c \in C$ to a platform $p \in P' \subset P$ described in a topology model I. It is defined as follows:

$$place : C \times P'^{+} \times I \rightarrow P$$

B. Decomposition Operator

The decomposition operator (*cf.*❶ on FIG. 3) is responsible for decomposing and deploying a data collection policy on a sensing infrastructure. This operator relies on the *Pre-deployment* process and the $deploy$ operator, both presented below.

1) Pre-deployment process:

a) Problem: Each concept involved in a data collection policy relies on data that come from one or different sensors and might only be deployed on a special kind of platform, *e.g.*, a collector concept requires a platform that is connected to the Internet. Therefore, before deploying the data collection policy over a sensing infrastructure, there is a need to identify the potential targets for each concept. This task can be assimilated to workflow enactment on computational grids [7].

b) Contribution: We define several operators that operate at the data collection policy level or at the infrastructure model level.

We introduce the req operator working at the data collection policy d level: for a concept $c \in concepts(d)$, the req operator returns the subset of sensors $S' \subset sensors(d)$ needed for the

[3]https://github.com/surli/tocsin – https://github.com/surli/spinefm

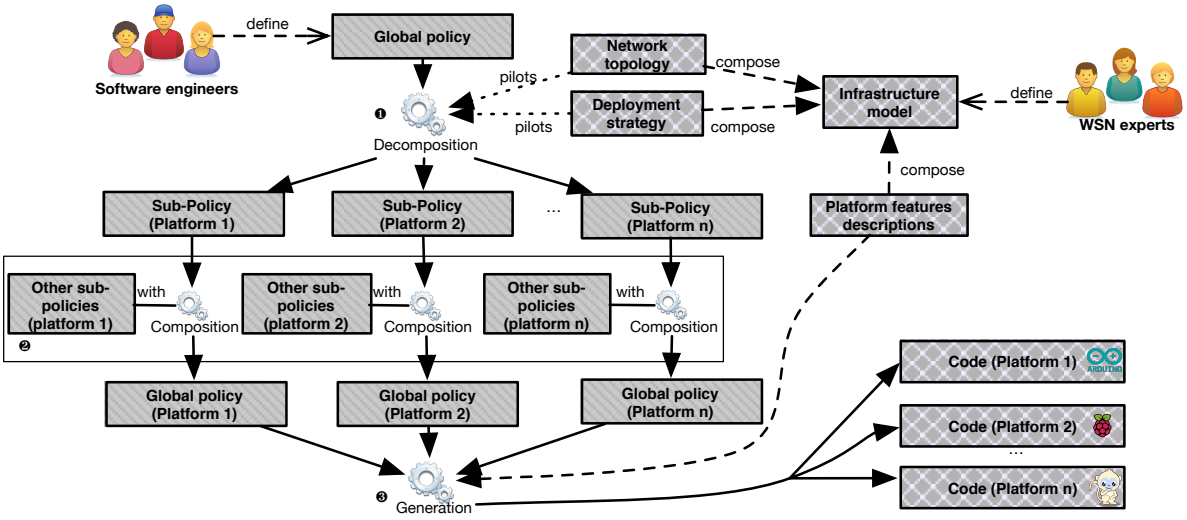

Fig. 3. Toolchain architecture overview

realization of the concept. The $isDeployable$ operator checks if the placing constraint equation for the concept c is satisfied on the platform p. The pre-deployment process is defined as follows: for a given concept c, the pre-deployment automation associates a property $targets$ that lists all the platforms $p \in platforms(i)$ that satisfy the following property: $req(c) \subset accessible(p) \wedge isDeployable(c, p)$.

c) Example: On FIG. 1, the *Filter* operation requires the *AC_443* sensor and data coming from this sensor can be obtained on *ARD_2_443*, *BRIDGE_1* and *ConnectedFaculty* platforms. This operation does not rely on hardware constraints, *i.e.*, $isDeployable(Filter, \_) = T$. The result of the Pre-deployment process is a *targets* property containing *ARD_1_443*, *BRIDGE_1* and *ConnectedFaculty*.

2) Deployment operator:

a) Problem: The pre-deployment process has identified several platforms for each concept. The sensor network expert needs to place concepts on candidates according to the deployment strategy that has been defined for the sensing infrastructure.

b) Contribution: The $deploy$ operator associates each concept with a platform and builds sub-policies for each platform involved in the realization of the data collection policy, such as $deploy : D \times I \rightarrow D^*$. The $deploy$ operator is defined as follows: given a pre-deployed data collection policy d_{pre} over an infrastructure described by the model i and a deployment strategy s, the $deploy$ operator associates with each concept $c \in concepts(d_{pre})$ a property $target$ that contains the result of the deployment strategy's $place$ operator, *i.e.*, $c.target = s.place(c, c.targets, i)$. Then, for each platform involved in the realization of the data collection policy, *i.e.*, $P_{inv} = \bigcup_{\forall c \in concepts(d_{pre})} c.target$, a platform-specific sub-data collection policy containing all the concepts that are going to be deployed on the platform (with their respective data-flow) is built. If the $place$ operator returns $\varnothing$ the deployment is aborted and an alert is sent to the software engineer that the concept c cannot be deployed with respect to the strategy s.

c) Example: On FIG. 1, the data collection policy is projected on the sensing infrastructure with a *Closest to the sensors* strategy. The deployment operator calls the deployment strategy's $place$ operator to find the platform (among the platforms identified as a result of the pre-deploy process) that matches the strategy.

C. Sharing Service

To enable sharing sensing infrastructures, the toolchain provides a composition operator linked to a proxy acting as a *sharing service* (*cf.* ❷ on FIG. 3). The sharing service maintains for each platform of the sensing infrastructure a set of sub-policies corresponding to those that are currently running on the platform.

When a new sub-policy is going to be deployed on a platform p, the sharing service maps each platform of the sensing infrastructure to a set of policies currently running on a platform. When a new platform-specific sub-policy need to be deployed on a platform p, the sharing service handles the composition of the new sub-policy with the previous deployed ones producing a *global sub-policy*.

D. Code Generation

The generation process (*cf.*❸ on FIG. 3) calls code generators to transform each platform-specific global sub-policy into executable code. One of the main issues to tackle here is to handle the variability existing between the different elements that compose a platform. The appropriate code generator γ for a platform p is selected according to the *Programming language* Platform Feature Description Model associated to p. The code generator γ has the responsibility to determine the order of generation of each concept and to use the right

libraries. To determine the order of generation, γ builds a list Π of constraints between concepts:

$$\forall f \in flows(d) \Rightarrow f.source < f.destination = \Pi$$

The constraints Π are then solved by a third-party constraint solver (in our case, CHOCO) giving the order of concepts generation. The execution of generation produces executable code for each concept and associate the adequate libraries to sensors and communication functions according the features described in the Platform Features Description Model. It must to be noted that platforms not involved in the realization of the data collection policy are flashed with a relay policy forwarding data inside the sensing infrastructure.

IV. EXPERIMENTS

The toolchain prototype named DEPOSIT[4] is available on Github. To evaluate the performance of the toolchain, we have designed and deployed several data collection policies on a simulated sensing infrastructure. The experiments were conducted on a mid-range laptop: 2.5 GHz Intel i7 (quad-core) 16 GB RAM. In the following, we consider 15 minutes as the reference time required by a sensor network expert to write in a low-level programming language the code implementing the example data collection policy for a single office without using any aspect of the toolchain.

We first validate the requirements presented in SEC. I by deploying a data collection policy at the scale of a building. Then, we scale up to a campus infrastructure.

A. Deploying a Smart Building Data Collection Policy

Defining a data collection policy: In this experiment, a software engineer is aiming to define the data collection policy presented in SEC. II. Each office is equipped with two *hall-effect* sensors (used to monitor the door and window positioning and two *temperature* sensors (used to monitor the ambient temperature and the air conditioning status). To simplify the definition of the data collection policy, the software engineer writes a *template* data collection policy that will be instantiated for each office. As a matter of practicality, the DEPOSIT source codes are presented on the companion web page. To validate our first requirement *(R_1) "a software engineer handles only business concerns"*, we consider that two constraints must be met: *a policy can be designed by only using business concepts* (C_1) and *a policy can be deployed without a priori knowledge* (C_2). As the software engineer handles only high-level concepts to express her needs, C_1 is obtained by design. The reuse of the infrastructure model defined by a sensor network expert abstracts the underlying infrastructure encapsulates the underlying complexity of the sensing infrastructure. This separation of concerns validates C_2, and therefore requirement R_1 is met.

Deploying the data collection policy: The software engineer reuses the office template to deploy a comprehensive data collection policy at the scale of 50 offices. As the sensing infrastructure is heterogeneous and multi-staged, the data collection policy must adapt to the specificities of the infrastructure (R_2) before being automatically projected (R_3).

To validate R_2, we consider that we need to satisfy that *the code resulting from the generation should call the right libraries* (C_3). Using SPL techniques, the code generation of concepts uses the complex configuration of platforms described in the infrastructure model and maps each feature to a specific library. As an example, the generation of the code reading and sending the temperature on the platform described on FIG. 2 uses the *Wiring* code generator. The calls to the temperature sensor use the `GroveTemperature.h` library. Finally, the sending of the value is performed by using the *ZigBee* library. This validates (C_3) and thus R_2.

To validate R_3, we consider that *each concept must be projected automatically on the sensing infrastructure* (C_4) and that *the code generation process must provide ready-to-flash code for each platform required by the data collection policy* (C_5). The deployment strategies defined by the sensor network expert determines the location of each concept. Then, code generators produce the source code for each platform. At small scale, we have deployed the source code generated from the approach on real hardware to ensure their functional validity. Then, we have scaled up to a larger deployment by simulating a smart building of 50 offices. As shown in TAB. I, DEPOSIT has generated a source code file for each platform involved in the realization of the global policy satisfying both C_4 and C_5. This finally validates our third requirement R_3.

B. Deploying Smart Campus Scenarios on a Shared Sensing Infrastructure

To validate our last requirement R_4, we simulate a smart campus infrastructure (based on a realistic sensor network deployment described as a reference architecture [6]) on which we deploy three data collection policies:

- P_1 *Air conditioning warning.* This data collection policy raises an alert when door and window are opened while the air conditioning is powered on (*cf.* SEC. II).
- P_2 *Fahrenheit converter.* The deployed temperature sensors return values in degrees Celsius. This policy allows the conversion to be done within the sensing infrastructure and a remote application to collect values in degrees Fahrenheit.
- P_3 *Monitor parking spaces occupancy.* This policy allows a remote application to know which parking spaces are available or occupied.

Our experimental setup is a *blank* sensing infrastructure (*i.e.*, no deployed data collections) deployed at the scale of 100 offices (each with 3 sensors) and 250 parking spaces (each with 1 sensor). To ensure data routing within the sensing infrastructure, we introduce 8 relay platforms and 1 border-router (bridging the sensor network to the Internet). A sharing service, as presented in SEC. III, handles the deployment of the data collection policies.

As the network is *blank*, the deployment of P_1 does not trigger any composition with other policies. However, as expected, the deployment of P_2 triggers 101 compositions. Indeed, as both P_1 and P_2 exploit the same platforms located

[4]Data collEction POlicies for Sensing InfrasTructures – https://github.com/ace-design/DEPOSIT

TABLE I
DEPLOYMENT OF THE RUNNING EXAMPLE (COMPREHENSIVE POLICY: 50 OFFICES)

	DEPOSIT source	# Generated files	# Generated LoC	# Concepts (before expansion)	# Concepts (after expansion)	Deployment time (in s)
Template	19	N/A	N/A	N/A	N/A	N/A
Single office	19	3	267	5	8	2.5
Comprehensive policy (without composition)	455	105	11685	250	400	50
Comprehensive policy (with composition)	$19 + 1$	105	11685	250	400	52

in offices and the same border-router, the sharing service needs to compose each sub-policy deployed on these platforms with the new ones. Finally, the deployment of P_3 triggers only a single composition on the border-router, as this policy only involves the parking sensors (that have not been yet exploited), and the border-router (that hosts the product of the composition of a sub-policy of P_1 and a sub-policy of P_2).

This experiment shows that the sharing service handles the deployment of multiple applications successfully, validating our last requirement R_4.

C. Threats to Validity

Testing infrastructure: Our approach has been validated both at small scale (~ 10 platforms) on real hardware (Arduino and Raspberry based platforms) and at large scale on a simulated infrastructure allowing us to vary parameters such as the number of sensors and platforms and the configuration of each. In order to be closest to the existing infrastructures, we have simulated the SmartCampus [6] infrastructure for office platforms and the SmartSantander[5] infrastructure for parking platforms. As part of future work, we plan to exploit a real large scale sensing infrastructure and study how the toolchain satisfy the needs of software engineers.

Deployment of new policies: The proposed toolchain handles the composition of data collection policies and code generation. The software engineer and sensor network expert obtain as a result a set of ready-to-flash source files. However, we do not perform a dynamic deployment over all sensing infrastructures as some platforms need to be re-flashed with a new firmware.

Prototype: The toolchain implementation is only a prototype, but it is publicly available and is verified by a large set of unit and integration tests. Still the presented results could have been disrupted by some bugs, but we believe they are close enough from what was expected to consider remaining bugs as not detrimental to results.

V. RELATED WORK

Related case studies: During the development of our toolchain, we have studied recent European projects to reuse their scenarios [26], [1]. All the described scenarios bring a need for data collection policies to build new innovative services on top of sensing infrastructures. However, the deployed platforms are flashed with ad-hoc code making them domain-specific and tailored for a particular application [16]. Our toolchain addresses this issue by allowing the sharing of the infrastructure between different applications and by producing code resulting from the composition of the deployed applications.

Modeling of data collection policies: In the process of developing applications for wireless sensor networks, Tei et al. [28] propose a *"stepwise software developement process for average developers"*. They identify two types of concerns: network related concerns and data-processing concerns. In order to assist software engineers, they provide a support to reuse network related solutions designed by sensor network experts. However, they do not provide a framework that allows the reuse of an infrastructure as we do with our toolchain.

Low-level data collection support: Most of the time the handling of the hardware heterogeneity in the IoT is performed by using embedded operating systems such as Contiki [9] or TinyOS [18]. They bring to the developer a hardware abstraction that make a program reusable over different platforms supporting the same OS. If the programming language supported by these OSes encapsulates the specificities of platforms into programming primitives, it remains oriented for sensor network experts as the developer still needs to manually address memory and energy concerns, as well as network communications. This last point also presupposes a deep knowledge of the network infrastructure.

Development toolchains for the IoT: The multi-stage model-driven approach for IoT application development proposed by Patel et al. [21] focuses on a clear identification and separation of the different stakeholders involved in the development process: *domain experts*, *application designer*, *application developer*, *network manager* and *device developer*. Each of them provides an input specification according to their expertise field. Several operators then manipulate these inputs to produce the application. Users in our approach are similar as we can assimilate application designers and application developers as software engineers and network managers and device developers as sensor network experts. However, we differ in the resulting application. They focus on building user-centric application with sensors and actuators whereas we only concentrate on the definition and the deployment of data collection policies for third party applications. More importantly, they do not support the sharing of the sensing infrastructure between several applications.

Generating embedded software: Portocarrero et al [24] introduce a process that manages wireless sensor networks in

[5]http://www.libelium.com/smart_santander_smart_parking/

an automatic way according to context changes, application requirements and adaptation policies. Their approach is interesting as they involve *Software architects*, *Domain experts*, *WSN experts* and *WSN developers* in the process. They make the assumption that *software architects (...) often do not have the knowledge about programming, especially on coding for WSN platforms*. To allow software architects to develop for WSN platforms, they introduce model transformations to map an *"architecture instance into platform-specific code"*. If the approach seems similar to ours, they only focus on Contiki-based platform while we provide a platform-independent toolchain that exploits variability models for its code generator to adapt to potentially any kind of platform currently available.

Automated toolchains: The approach proposed by Di Cosmo et al. [8] relies on a toolchain enabling the development of complex distributed applications with a high level of abstraction. Their toolchain respects the separation of concerns paradigm by taking several inputs: a high-level specification of their application (similar as our data collection policies), a description of the system (similar as our infrastructure model) and a description of the current system. We differ on this last point by providing a sharing service that abstracts the current state of the sensing infrastructure for a software engineer. The deployment phase of their toolchain assumes that all their targeted VMs are empty. In our approach, we support addition of new policies and we ensure that a new data collection policy is composed with the previous deployed ones.

VI. Conclusion

In this paper, we have presented a toolchain that allows software engineer to express and deploy data collection policies over heterogeneous shared sensing infrastructure. The toolchain relies on the separation of concerns principle where software engineers only deal with business concerns and sensor network experts provide reusable infrastructure models. Data collection policies expressed by the software engineers are decomposed on the sensing infrastructures and composed with the previous deployed ones. This supports the sharing of infrastructures and enables reuse of data collection policies. The code generation resulting from the decomposition is tailored to the different platforms according to the infrastructure model expressed by a sensor network expert, thus taming the diversity and heterogeneity of sensing infrastructures.

We have successfully modeled and deployed data collection policies on simulated but realistic sensing infrastructures and showed that the sharing operations are performed automatically. Future work will first aim at collaborating with an industrial partner to assess the benefits of our approach on large scale sensing infrastructures. We also plan to work with software engineers in order to get feedback about the definition of data collection policies. We finally plan to introduce a new layer of abstraction that would allow domain experts to interact directly with the toolchain.

References

[1] Butler project. http://www.iot-butler.eu/.

[2] F. Anon, V. Navarathinarasah, M. Hoang, and C.-H. Lung. Building a framework for internet of things and cloud computing. In *Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing (CPSCom), IEEE*. IEEE, 2014.

[3] S. Apel, D. Batory, C. Kästner, and G. Saake. *Feature-oriented software product lines: concepts and implementation*. Springer Science & Business Media, 2013.

[4] S. Apel and C. Kästner. An overview of feature-oriented software development. *Journal of Object Technology (JOT)*, 8(5), 2009.

[5] D. Benavides, S. Segura, and A. Ruiz-Cortes. Automated analysis of feature models 20 years later: a literature review. *Information Systems*, 35(6), 2010.

[6] C. Cecchinel, M. Jimenez, S. Mosser, and M. Riveill. An Architecture to Support the Collection of Big Data in the Internet of Things. In *SERVICES'14*. IEEE, 2014.

[7] E. Deelman et al. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1(1), 2003.

[8] R. Di Cosmo et al. Automated synthesis and deployment of cloud applications. In *ASE'14*, pages 211–222. ACM, 2014.

[9] A. Dunkels, B. Grönvall, and T. Voigt. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *LCN'04*, pages 455–462. IEEE, 2004.

[10] G. Fuchs and R. German. Uml2 activity diagram based programming of wireless sensor networks. In *SESENA'10*. ACM, 2010.

[11] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Generation Comp. Syst.*, 29(7), 2013.

[12] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7), 2013.

[13] W. L. Hürsch and C. V. Lopes. Separation of concerns. 1995.

[14] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA). Technical Report CMU/SEI-90-TR-21, 1990.

[15] H. Karl and A. Willig. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, 2005.

[16] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos. Wireless sensor network virtualization: early architecture and research perspectives. *Network, IEEE*, 29(3), 2015.

[17] P. Kruchten. *The rational unified process: an introduction*. Addison-Wesley Professional, 2004.

[18] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. Tinyos: An operating system for sensor networks. In *Ambient intelligence*. 2005.

[19] F. Li, M. Vogler, M. Claeßens, and S. Dustdar. Towards automated iot application deployment by a cloud-based approach. In *SOCA'13*.

[20] G. Mainland, G. Morrisett, and M. Welsh. Flask: Staged functional programming for sensor networks. In *ACM Sigplan Notices*, volume 43. ACM, 2008.

[21] P. Patel and D. Cassou. Enabling high-level application development for the internet of things. *Journal of Systems and Software*, 103, 2015.

[22] A. Paulon, A. A. Frohlich, L. B. Becker, and F. P. Basso. Wireless sensor network uml profile to support model-driven development. In *INDIN'14*. IEEE, 2014.

[23] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, 2005.

[24] J. M. Portocarrero, F. C. Delicato, P. F. Pires, T. C. Rodriges, and T. V. Bastista. Samson: Self-adaptive middleware for wireless sensor networks. In *SAC'16*. ACM, 2016.

[25] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava. Energy-aware wireless microsensor networks. *Signal Processing Magazine, IEEE*, 19(2), 2002.

[26] L. Sanchez et al. Smartsantander: The meeting point between future internet research and experimentation and the smart cities. In *Future Network Mobile Summit (FutureNetw), 2011*, 2011.

[27] S. Schmid and R. Wattenhofer. Modeling sensor networks. *Algorithms and Protocols for Wireless Sensor Networks*, 62, 2008.

[28] K. Tei et al. Model-driven-development-based stepwise software development process for wireless sensor networks. *Systems, Man, and Cybernetics: Systems, IEEE Transactions on*, 45(4), 2015.

[29] S. Urli, M. Blay-Fornarino, and P. Collet. Handling complex configurations in software product lines: a tooled approach. In *SPLC'14*.

[30] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *INFOCOM'12*, volume 3. IEEE, 2002.