Leveraging Declarations over the Lifecycle of Large-Scale Sensor Applications

Milan Kabáč, Charles Consel, Nic Volanschi

Leveraging Declarations over the Lifecycle of Large-Scale Sensor Applications

Milan Kabáč
Inria Bordeaux
Bordeaux, France
email: milan.kabac@inria.fr

Charles Consel
Bordeaux Institute of Technology
Bordeaux, France
email: charles.consel@inria.fr

Nic Volanschi
Inria Bordeaux
Bordeaux, France
email: eugene.volanschi@inria.fr

***Abstract*—Masses of sensors and actuators are being deployed in our daily environments to provide innovative services for such spaces as parking lots, buildings, and railway networks. Yet, to realize the full potentials of these sensor network infrastructures, services need to be developed. Service development raises a number of challenges due to existing approaches that are often low level and network/hardware-centric. This paper proposes a high-level approach to the development of large-scale orchestrating applications. It revolves around a declaration language that allows to express the sensor-network dimensions of an application (sensor discovery, delivery models, actuation process). These declarations define the behavior of an application with respect to the sensor network infrastructure. We demonstrate the key relevance of these declarations at every stage of an application lifecycle, from design to runtime. In doing so, declarations allow to match the sensor-network behavior of an application to the target infrastructure. Our approach summarizes and puts in perspective our development of industrial case studies and our experience in using a commercially-operated sensor infrastructure.**

I. INTRODUCTION

Infrastructures of masses of sensors are increasingly emerging in our environment and being deployed over large-scale spaces, including parking lots in cities and agricultural fields in rural areas. These large-scale infrastructures are now being operated worldwide by companies, enabling economically viable services to be offered. For instance, the Smart City project in Santander provides an infrastructure composed of sensors, actuators, cameras and screens to monitor available parking spaces, offering valuable information to car drivers [1]. Similarly, Sigfox [2] deployed a large-scale infrastructure in Moscow, providing the city with the world's largest intelligent parking system, comprising fifteen thousand sensors; this system has been operated since 2014 [3].

Even though, these deployments demonstrate the maturity and practicality of such infrastructures, there are still challenges that need to be addressed to harness the potentials of this technology. In particular, the process of developing services for masses of sensors is still ad hoc. Current practices are driven by the network operator (*e.g.,* Sigfox) and centered around the concerns of the specific stakeholders: sensor manufacturers. Moreover, research in the domain often ignores realistic application-specific requirements as discussed by Raman *et al.* [4]. This network/hardware-centered approach makes software development low level and feature specific, resulting in a steep learning curve for programmers. This situation can be a major impediment for the success of the domain.

Just like mobile application development, large-scale sensor infrastructures need to provide programmers with methodologies and tools to support the development of services. This work is essential to allow innovative services to be developed, leading to the adoption of such infrastructures. For example, the Android platform comes with a software framework that manages the lifecycle of applications, provides access to device resources, and allows data sharing between applications. In addition, developers need to make explicit, via a manifest file, a number of application properties (*i.e.,* application components, permissions, *etc.*). This manifest provides a high-level view of an application, which is leveraged by the Android device owner, as well as the operator of the Android infrastructure (*i.e.,* Google), to reduce the risk of abusing resources (*e.g.,* privacy, battery, network) when running this application.

Similarly, for large-scale sensor infrastructures, an application would need to expose, prior to runtime, how it may use resources. Specifically,

- What are the required sensors/actuators?
- How does it gather/receive data from sensors?
- When are sensor readings delivered?

Such issues need to be examined at various stages of an application lifecycle, from design to runtime, and may raise such infrastructure concerns as determining whether the sensor infrastructure can provide the application with the required resources (*i.e.,* admission control), whether the infrastructure needs to be configured to factorize sensor readings (*i.e.,* caching mechanism). Domain expertise is needed to examine infrastructure concerns along the lifecycle of the application. Currently, this expertise is implicit, making the few existing experts a bottleneck for producing new services that take into account the infrastructure concerns. As a result, not only are sensor-network characteristics of an application missing to support its development, but they are also missing to match the infrastructure to the application needs.

This paper

We propose an approach that makes explicit the domain expertise required to guide the development and deployment of a large-scale orchestrating application. To do so, based on the literature and practical insights, we introduce the notion of *application behavior*, which characterizes how an application behaves to orchestrate sensors at a large scale along three dimensions: service discovery – what sensors are required, data delivery – how and when data are delivered to the application, and actuating process – what actuators are issued orders by the application. These behavior dimensions are then instantiated across the stages of the application lifecycle and examined with respect to sensor infrastructure concerns. This process ranges from checking that the sensing capabilities required by an application at design time are compatible with the target infrastructure, to submitting an application to an admission control procedure at deployment time. Our contributions can be summarized as follows.

1) ***Application behavior.*** We identify the sensor-network characteristics of a large-scale orchestrating application. As such, the characteristics of an application can be expressed at a high level and early in the development process, providing support throughout the application lifecycle.
2) ***Declaration language.*** We present a simple declaration language that allows to express the key sensor-network behavior of an orchestrating application. We show how declarations make explicit essential information about the application.
3) ***Application lifecycle.*** We introduce stages, along the application lifecycle, where the behavior declarations of an application can be used to adapt both the application and the infrastructure concerns.
4) ***Validation.*** We validate our approach with a real-sized case study, namely, a citywide parking management system. This case study illustrates all the aspects of our proposed approach and demonstrates how it addresses the sensor-network behavior of an application.

We build on practical experience gained from (1) working with operators of sensor-network infrastructures [5] and (2) previous implementations of case studies leveraging these infrastructures [6], [7]. As such, our approach provides a design framework that paves the way for researchers working on methodologies and tools supporting the development of applications for large-scale sensor infrastructures.

Outline

The next section presents our case study, used throughout the paper. Then, Section III decomposes our notion of application behavior into a set of key sensor-network dimensions, drawn from the literature on sensor networks. These dimensions give a design framework for a declaration language dedicated to the sensor-network behavior of applications. This language is introduced in Section IV, as well as the main stages of an application lifecycle. Section V discusses how our approach can leverage existing approaches addressing infrastructure concerns. Related works are covered in Section VI and concluding remarks are given in Section VII.

II. CASE STUDY

In this section, we present a case study used throughout the paper to introduce the salient aspects of our work. This case study examines the development of a parking management application, inspired by existing smart city projects mentioned earlier. The purpose of the application is to monitor the occupancy of parking spaces and regulate the flow of traffic by directing cars to available parking spaces. Presence sensors are buried under the ground and determine the occupancy of parking spaces by measuring magnetic field variations. The application also requires Carbon Monoxide (CO) sensors to detect an unsafe level of pollution in parking lots, and alert drivers if this situation occurs. Parking availability and pollution alerts are displayed on display panels at the entrance and inside the parking lots.

The goal of this application imposes a number of requirements on the process of orchestrating masses of sensors and actuators.

Req 1. Both presence and CO sensors need to be grouped by parking lots and levels. Indeed, space availability and pollution levels are naturally delivered at both these granularities by a parking management system.

Req 2. Presence at a parking space is published by the associated sensor when its status changes; an event-based delivery model is well-suited for a boolean type of sensor.

Req 3. CO sensors deliver their measurements in two ways depending on the needs of the data consumer. For the ventilation of the parking lot, the information is produced every 15 minutes; this time may vary depending on the time it takes to renew the air for a ventilation system and the size of the parking lot. For the purpose of pollution alert, we require the pollution level to be delivered when it reaches a given threshold.

In the remainder of the paper, we address these requirements at different stages of the development lifecycle of the application. From a general perspective, requirements on orchestrating sensors and actuators are identified along three main dimensions, presented in the next section.

III. APPLICATION BEHAVIOR DIMENSIONS

A sensor network is an environment constrained in many ways (bandwidth, energy, computational power, *etc.*). When this environment serves a resource-intensive application or resource-competing applications, their usage profile needs to be determined to ensure quality of service. Towards addressing this issue, we introduce the notion of *application*

behavior dedicated to sensor-network dimensions. In this section, we leverage the literature on sensor networks and decompose the notion of application behavior into three key dimensions.

A. Service Discovery

Service discovery for large-scale orchestrating applications poses unique challenges due to the resource-constrained nature of sensor networks. Service discovery defines the sensor nodes of interest, the *sources*, and the applications consuming sensor data, the *sinks*. The more accurately sources are selected by applications, the less communication occurs between sources and sinks. This strategy is of utmost importance in the context of a bandwidth-poor environment, comprising masses of sensors.

Meshkova *et al.* [8] notice that sensor nodes with limited computational resources are not suited for computational and memory hungry service discovery protocols. Furthermore, most well-known protocols, such as UPnP or SLP, are too large to be processed by a sensor network. A promising approach to service discovery, proposed by Heidemann [9], is to organize sensor-network communications with respect to the application *attributes*, rather than with respect to the network topology.

Estrin *et al.* [10] notice that this application-specific approach is aligned with common application scenarios in the domain. That is, it relies on data generated from the sensor network infrastructure, rather than from individual sensors. Concretely, applications are more likely to ask: *"Where are the nodes whose temperature recently exceeded 30 degrees?"* than *"What is the temperature at sensor #27?"*.

Heidemann *et al.* [9] also demonstrate the benefits of an attribute-based naming approach, when driven by application-specific requirements. In particular, they show that this approach significantly reduces network traffic.

Based on these works, we conclude that the service discovery dimension should be made explicit and contribute to the requirements of an application, early in the development process. This service discovery dimension should consist of attributes (*e.g.,* sensor types and locations), exposing information to optimize the target sensor network (*e.g.,* network traffic).

B. Data Delivery

Beyond the service discovery dimension, an application behavior is also characterized with respect to how sensor data are delivered. Tilak *et al.* [11] propose a classification that introduces fundamental delivery models: continuous, event-driven, observer-initiated and hybrid. Importantly, this classification is defined with respect to the observer's interest (*e.g.,* the application). The data delivery model of an application needs to be explicit to avoid mismatches between application requirements and the target sensor network infrastructure. For example, an application may require data to be delivered at a certain frequency. However, this requirement may not be fulfilled because the target infrastructure does not provide enough bandwidth to transmit the data. Similarly, an application may need to access sensors in a query-driven fashion, which may not be supported by the target infrastructure.

It is important to notice that the data delivery models required by applications also suggest a network structure and specific algorithms that best match the applications' needs, especially in the context of a resource-poor environment. For instance, in her Ph.D. thesis, Heinzelman showed that clustering is most efficient for static networks, where data is continuously transmitted [12].

The number of sources and sinks used by an application is also valuable information for choosing an appropriate communication strategy [13]. Liu *et al.* [14] introduce a communication pattern for large-scale sensor networks that adapts the communication strategy with respect to the relative frequency between application queries and detected events from sensors. Furthermore, because dissemination protocols for large-scale sensor networks are data-centric, they can exploit application semantics to improve performance [15]. Communication costs can be reduced by introducing some computation inside the network; this is referred to as in-network data aggregation and processing [9]. For example, information on how data is to be presented to an application (*e.g.,* parking spaces via parking lots) can be used by the sensor network infrastructure to perform in-network data aggregation per node cluster (*i.e.,* parking lots). Estrin *et al.* carry out this idea by using intermediate nodes to perform application-specific data aggregation and caching [10]. *Localized algorithms* are introduced in the context of distributed computations to allow sensors to only communicate with neighbor sensor nodes. As the number of nodes increases, localized clustering can contribute to more scalable behavior, since communication between nodes is kept within a neighborhood.

To summarize, researchers have shown that application-specific information is critical to optimize sensor networks at various levels. This application-specific information mainly consists of knowing what sensors are used by an application and how sensor data are to be delivered. That is, service discovery and data delivery.

C. Actuating

When orchestrating devices in the large, the nature of actuators introduces some differences in the way an application uses them, compared to sensors. In our experience, we identify two approaches to issuing orders to actuators. The first approach is symmetrical to sensors: it consists of *multicasting* an order to a group of actuators. In our case study, such operation is used to inform car drivers of parking availability at the periphery of the city. The second approach to issuing orders to actuators corresponds to what

is done when orchestrating devices in the small; it consists of invoking a specific actuator (*e.g.,* an information display at a given parking level) to perform a context-specific action (*e.g.,* display the number of available parking spaces of a given level).

Exposing how an application invokes actuators provides the sensor network infrastructure with valuable information. For example, individually invoking actuators is likely to be a key outcome of an application. Therefore, the application should probably not be launched, if individual actuators are not reachable, or at least, human intervention should be required to resolve the problem. In contrast, when an application invokes actuators using multicast, it should have some impact, even though some actuators may not be reachable when launching it.

IV. STAGES OF APPLICATION LIFECYCLE

To support orchestration in the large, we introduce declarations expressing the sensor-network dimensions of an application. To address these dimensions systematically, they are matched against each stage of the application lifecycle, from design to runtime. This approach has the following methodological and programming benefits.

- Sensor-network dimensions of an application are expressed early, and gradually matched against the stages of the application lifecycle.
- This gradual mapping raises a need for adaptation layers, when the sensor-network dimensions of an application cannot be reconciled with the target sensor network infrastructure (*e.g.,* missing sensors, unavailable delivery model).

As can be noticed, our staged approach keeps the application development independent from infrastructure details (*e.g.,* network protocol, bandwidth, data link features). In doing so, we strive to be as agnostic to the network as possible. We present the different stages of the application lifecycle and instantiate each stage with our case study.

In this paper, we do not provide a formal definition of our declaration language dedicated to sensor-network dimensions of an application. Because it is simple, this language is presented informally, throughout the next section, with fragments from our case study. The information denoted by these fragments, declared at design time, is leveraged across the remaining sections devoted to the later stages.

A. Design stage

The declared sensor-network dimensions provide the blueprint for later stages in the application lifecycle. In particular, they are valuable documentation to be used by the stakeholders of the sensor network infrastructure. Let us illustrate our approach with our case study of parking management, following the behavior dimensions introduced previously.

Discovery of sensors and actuators. Orchestration in the large requires to categorize the sensors and actuators of interest. To do so, the designer needs to leverage a physical-space partitioning that is well-suited for their target application. This partitioning is likely to have been introduced by the infrastructure owner and used to register sensors and actuators as they get installed. A partitioning expressed in terms of declarations is given in Fig. 1; it fulfills Requirement 1 of our case study (see Section II). In these declarations, parking spaces are members of parking levels, which are members of parking lots. This partitioning of spaces can then be used by an application to discover sensors grouped by parking lots, matching the granularity most common to car drivers. Last, devices are registered as being in a city entrance, if they are so located. Fig. 2 uses these partitioning declarations to define application-specific discovery for sensors and actuators. For each parking lot, the application discovers presence sensors (Lines 4 to 7) and CO sensors (Lines 9 to 12) that are populating each parking level. Information displays are discovered within the parking lot for user information (Lines 14 to 17) and at the periphery of the city to guide car drivers (Lines 19 to 22).

Data delivery. The designer needs to define how data are delivered to the application. Fig. 3 presents delivery model declarations for sensors, which build upon service discovery rules introduced previously. Event-driven delivery model is chosen for presence sensors because it allows to react only when the status of parking spaces changes (Line 1). Given that this status is a boolean value, events are the most natural

```
1 parking_level INCLUDES parking_space
2 parking_lot INCLUDES parking_level
3 city INCLUDES city_entrance
```

Figure 1. Extracts from the city partitioning of spaces.

```
1  APPLICATION parking_management
2  IMPORT city_partitioning

4  DISCOVER presence_sensors = {
5    service = sensor
6    source = presence :: boolean
7    group = parking_lot }
8
9  DISCOVER co_sensors = {
10   service = sensor
11   source = co_level :: float
12   group = parking_lot }
13
14 DISCOVER plot_info_displays = {
15   service = actuator
16   action = display
17   group = parking_lot }
18
19 DISCOVER city_info_displays = {
20   service = actuator
21   action = display
22   group = city_entrance }
```

Figure 2. Application-specific service discovery for the parking management application.

```
1 DELIVERY presence_sensors AS event_driven
2 DELIVERY co_sensors AS periodic::15::min AND event_driven
```

Figure 3. Application-specific data delivery for the parking management application.

delivery model. This fulfills Requirement 2 of our case study. To fulfill Requirement 3, CO sensors are assigned the periodic data delivery model and the event-based model (Line 2). The periodic model is used by the ventilating system to renew the parking air, whereas the event-based model is used to warn users of an air pollution event when the CO level has reached a given threshold.

Actuating. Fig. 4 presents declarations for actuating information displays using service discovery rules introduced earlier. Information displays are actuated with respect to their level within the parking lot since the application tailors the information to this granularity (Lines 1 to 2). However, at the city periphery, information displays are actuated more globally because they receive recommendations for drivers entering the city and looking for an available parking space at their destination (Lines 4 to 5). In our case study, we envision information displays performing some local processing to select relevant information (*e.g.,* parking lots nearby a city entrance). For this purpose, we use the `MULTICAST` directive that triggers a partition of information displays (*e.g.,* at city entrances) to disseminate information on parking lots. A `FOREACH` directive is also available to allow for fine-grained actuation of ventilation systems based on the CO level measured at each parking level (not shown in this paper). Despite being high-level, our declarations provide a precise conceptual framework for the programming stage.

B. Programming stage

This stage consists of developing the application, addressing all of its sensor-network dimensions that involves programming. In doing so, data structures accommodating sensor readings are implemented. Furthermore, the processing of sensor readings is programmed with respect to data delivery models, chosen at design time (Fig. 3). Finally, note that the approach to processing data and producing results typically follows what has been defined at the design stage. For example, the reporting on parking space availability follows the granularity of the service discovery (*i.e.,* parking levels) (Fig. 2). Furthermore, the processing of presence sensor readings follows an event-based model. Also, design declarations result in software architectural patterns. For example, a callback mechanism is typically implemented to serve an event-driven delivery model such as the one declared for presence sensors (Fig. 3, Line 1).

C. Deployment stage

At deployment time, the key features of the target sensor network infrastructure are revealed. These features should thus be matched against the sensor-network requirements of

```
1 TRIGGER display ON plot_info_displays
2   MULTICAST parking_level
3
4 TRIGGER display ON city_info_displays
5   MULTICAST city_entrance
```

Figure 4. Application-specific device actuation for the parking management application.

the application to identify whether adaptations are needed. Let us illustrate this stage in the context of our parking management application. For service discovery, we need to verify that the target sensor network infrastructure provides the required sensors and actuators declared in Fig. 2. Assume that CO sensors are not present, the deployment process should fail. Alternatively, if regulation does not require CO sensors, an adaptation layer could be invoked to produce fake values. Such a situation would likely demand human intervention.

Similarly, if bandwidth limitations in the sensor infrastructure or energy constraints on a sensor type prevents the periodic delivery requirement to be fulfilled (*e.g.,* Fig. 3, Line 2), a human intervention may be needed to make a decision. Such a situation would occur, for example, if the periodicity of CO measurements needed to be adapted because it likely needs to adhere to strict regulations.

The partitioning of sensors is another key feature of the target infrastructure. Adaptation code may be required to map the infrastructure partitioning into the application partitioning; this can be done when the application partitioning is more coarse than the infrastructure partitioning. For example, if the application required presence sensors to be grouped per parking lot (Fig. 2, Line 7) and the infrastructure only grouped them per level, a layer could gather the sensors for all levels to deliver the appropriate information to the application. For data delivery, the models of the target infrastructure are matched against the ones required by the application. Some adaptation strategies can be used to account for mismatches. For example, an event delivery model can be simulated with periodic delivery, combined with an event condition. In our case study, if an application alerts parking users when the air pollution has reached a given threshold, it might select an event-based delivery model to achieve this goal. An adaptation layer consists of monitoring the CO periodic measurements and trigger an event when a given threshold is reached. In practice, depending on the physical architecture of the sensor network infrastructure, adaptation code may be placed close to the sensors (*e.g.,* at a base station) or run as an additional layer of the application if the infrastructure cannot be extended.

D. Launch stage

When launched, the application may not have access to its required resources, if they are already serving other applications, or if the infrastructure has been temporarily or permanently reconfigured. This stage is distinct from

the next stage, namely runtime because it does not address changes that may impact the application, while it is running. The aim of this stage is to adapt for changes that occurred between the time the application was deployed and its launching. For example, the sensor network infrastructure may have reserved some bandwidth to serve a given periodicity for CO sensors and be unable to fulfill this requirement because of a temporary failure of network nodes.

E. Runtime stage

At run time, the operating conditions of the application can change arbitrarily. For example, resources may be put offline for some technical reasons. In our case study, a base station failure may disconnect a number of sensors and actuators, sensors may fail, bandwidth may degrade, *etc.* These changing conditions can violate the quality of service requirements of the application and compromise its purpose. In our case study, if the failure of CO sensors does not offer the expected coverage of the parking lot levels, the threshold for a pollution level may not be reached because of the resulting inaccurate measurements. A crude approach could consist of terminating an application when its sensor-network requirements are violated. Human intervention could then be required to analyze the situation and resume operation, if possible. A more advanced solution would consist of introducing a runtime mechanism that monitors the cardinality of the sensor partitions defined by the application. This mechanism should allow the application to adapt at runtime when operating conditions degrade. Similarly, the application needs a mechanism to react to data delivery models that are violated at runtime. Interfacing these events with a programming language can be done via the exception mechanism by introducing exceptional events dedicated to runtime errors, as described by Mercadal *et al* [16], or violation of quality of service contracts, as presented by Gatti *et al.* [17].

Summary. Table I summarizes how declarations of sensor-network dimensions are leveraged throughout the application lifecycle. As can be noticed, the range of actions based on declarations is very large: from guiding programming to admission control at deployment time, to coordinating concurrent activations of actuators at runtime. The table illustrates the framework laid out by our approach, providing a spectrum of opportunities that goes beyond our case study.

V. DISCUSSION

We first explain how our work can leverage existing approaches on sensor networks. We then discuss a related development approach and discuss how it could be extended with the present work.

A. Leveraging other approaches

This section shows how our sensor-network declarations could be further exploited for adapting the infrastructure to the application needs. This could be done by leveraging various existing approaches to application-specific optimizations in sensors/actuators networks. The discussion is organized around a set of the key infrastructure concerns drawn from the literature on sensor networks.

1) Admission control: Madria *et al.* [18] propose the Sensor Cloud paradigm as a computing environment spread in a wide geographical area, unifying multiple WSNs, and available to one or multiple applications. This can be viewed as an extension to the notion of Cloud computing, adding virtualized sensing and actuating abstractions. The Missouri S&T Sensor Cloud is a concrete realization of this paradigm. It provides applications with sensing as a service, taking such parameters as the region of interest, the frequency and latency of sensed data. An admission control module (called provision management) examines the service requests to decide whether they can be fulfilled. When physical sensors are virtualized to several applications, this module computes the sampling durations and frequencies for satisfying all the requests. This configuration is recomputed when new applications join, or existing applications leave the system.

Because our approach exposes application needs, such as the required sampling frequencies of sensors, it could directly benefit to this paradigm, automating the service (re)negotiation between the application and the sensor Cloud, at different stages.

2) Network configuration: Heideman *et al.* [13] show how WSN application performance can be improved by up to 60% and network traffic cut by half, by matching data dissemination algorithms to the application requirements. To this purpose, they offer a network API, allowing the application developer to choose between several data diffusion algorithms, such as push-based or pull-based. Additionally, they provide experimental data to determine which algorithm is best for which application communication patterns. For instance, a pull-based algorithm (namely two-phase pull diffusion) performs poorly when there are many sensors potentially sending data to many sinks but the sensor data are actually sent rarely. In this situation, some push-based diffusion algorithms can significantly improve performance.

Clearly, by making network-sensor dimensions of applications explicit, our approach allows to automatically select the appropriate dissemination algorithm. The stage at which this selection should occur depends on when relevant information is known. In a single-application setting, where both sensor discovery and periodicity are known statically, the network can be configured at design time. For another example, if sensor periodicity is static but the number of actual sensors is known later, the choice can be done at deployment time or at launch time. In multi-application settings, the selection of a dissemination algorithm must at least be examined at each application launching.

Liu *et al.* [14] introduce a family of data dissemination/-gathering algorithms in $n \times n$ grid WSNs, spanning the

Table I
ADDRESSING SENSOR-NETWORK DIMENSIONS THROUGHOUT THE APPLICATION LIFECYCLE.

	Design stage	**Programming stage**	**Deployment stage**	**Launch stage**	**Runtime stage**
Goals	High-level specification	Supporting & guiding programming	Admission control	App/Network adaptations	Resolving QoS violations
Service discovery	Declaring sensors/actuators and their partitioning	Implementing data structures	Matching sensors/actuators	Validating resource allocation	Coping with infrastructure failures
Data delivery	Declaring data delivery models	Implementing data processing models	Matching data delivery	Validating & accommodating data delivery	
Actuating	Declaring actuation strategies	Implementing actuation strategies	–	Validating & accommodating actuation	Coordinating concurrent actuation

whole space between pure push – when sensors send data to applications, to pure pull — when applications send queries to sensors. Their algorithm family is based on a variable diffusion structure, similar to a comb. They show how the optimal balance can be expressed as a function of the grid size n and the relative frequencies of application queries and sensor events. As both frequencies are made explicit in our approach, the right push *vs* pull configuration in such a network could be computed automatically, as soon as the grid size n is known. For static networks, the configuration can be done at design time. When sensors are discovered at a later stage, such as deployment or launch time, the choice has to be performed at the corresponding stage.

Delicato *et al.* [19] propose an architecture based on web services, in which sensors and applications declare the services they provide, respectively need, using standard SOAP configuration messages. These service descriptions include the sensor and data type, the geographical location, the acquisition interval (data rate), and the acquisition duration. A threshold may also be specified for non-periodic sensing. These sensors and application characteristics are used during network configuration for setting up the data dissemination protocol to minimize the energy consumption of the sensors used for delivering their data to client applications. When using our approach, the application needs are made explicit at design time, and can be thus used to configure the network at deployment time or to reconfigure it at launch time, to accommodate an incoming application.

3) Event filtering and processing: TiNA [20] is a scheme for minimizing sensor power consumption by exploiting the temporal coherency tolerance of applications. This approach goes beyond in-network data aggregation in that it does not send sensor readings at all, if this fits the QoD (quality of data) needs of applications. Thus, applications express their sensing needs using annotated SQL-style queries. These queries mention the type of data, its possible aggregation, and periodicity, and are annotated with temporal coherency tolerance. TiNA uses these annotations to suppress sensor readings (hence to save power) while still providing high quality approximated answers to application queries. This approach is complementary to ours in that applications are able to declare their sensing needs as early as possible. These declarations can be directly used to leverage sophisticated underlying optimizations such as those provided by TiNA.

B. A domain-specific language approach

Kabáč and Consel propose a domain-specific language dedicated to the design of applications orchestrating sensors [6]. Design declarations are processed to support and guide the programmer using generative programming. This strategy allows to abstract over the characteristics of the sensor network. To cope with issues specific to orchestrating masses of sensors, they introduce a design language that is dedicated to this domain, allowing the developer to declare what an application does, prior to programming it. This design language, named DiaSwarm, consists of constructs dedicated to manipulating objects at a large scale. For example, it provides constructs to declare delivery models of sensors as well as aspects related to parallel data processing at design time [7]. They have developed a compiler for DiaSwarm that produces a programming framework customized with respect to a given DiaSwarm design. This present work was inspired by DiaSwarm but it is more general in that it is not dedicated to supporting the programming stage and covers more aspects with its declaration language (*e.g.,* actuators). Furthermore, we go beyond DiaSwarm in that we examine how sensor-network dimensions can be leveraged throughout the lifecycle of an application. In doing so, we bridge the gap between the application needs and the sensor-network concerns: we describe the details of their interactions and leverage the literature in sensor networks. Using these results, DiaSwarm could be further developed by extending its compiler to generate code that would address the stages and infrastructure concerns presented in our work. For example, adaptation layers could be generated automatically, either added to the application code or loaded in the sensor-network infrastructure.

VI. RELATED WORK

To identify the key sensor-network dimensions of an application, we already mentioned a range of works (Section III). In this section, we review works pertaining to other aspects related to sensor networks, not discussed earlier.

From WSN to SSN: Historically, Wireless Sensor Networks (WSNs) have been designed for a single application. As a result, they could be highly optimized in an application-specific manner at design time. This approach is well suited for small-scale networks but does not scale for networks of thousands of nodes. Indeed, at this scale, the cost of network deployment and maintenance becomes more important and return on investment is a key issue. Consequently, in recent years, research has been focusing on building *Shared Sensor Networks* (SSN), allowing several applications to run concurrently on the same sensor network infrastructure. Despite their benefits, SSNs are still in an early stage, compared to WSNs. A very recent survey of SSNs [21] identifies several specificities of SSNs, raising new challenges that must be addressed. Such challenges include dealing with the heterogeneity of the infrastructure, dealing with resource contentions, and optimizing sensor data flows for several applications at the same time. Heterogeneity imposes a looser coupling between the applications and the infrastructure. Dealing with resource contention requires exposing application needs early to ensure that a new application does not interrupt currently running ones. Optimizing sensor data flows also requires exposing early the sensor-network behaviors of the application to enable cross-application optimizations. While SSNs bring concrete solutions to these challenges, they do not (yet) offer tools for streamlining the development of SSN applications.

Our work proposes an approach towards resolving this development problem by exposing the sensor-network dimensions of an application early, at design time. These dimensions can then be exploited at different stages. For example, at deployment time, our declarations address such challenges as resource contention.

Middleware: Several works propose middleware solutions for supporting large-scale sensing and actuating applications. A general approach to this problem domain is the SwarmOS vision. In a white paper [22], the authors promote a middleware platform for developing applications at the frontier between a large-scale WSN and the Cloud. The goal of SwarmOS is to offer high-level services, intermediating between the WSN/Cloud resources and applications (called "swarmlets"). Such services include access control and virtualization, data summarization and aggregation, discovery, *etc.* While such middleware services could greatly simplify the development of sensing/actuating applications, they do not offer a systematic method for developing such applications. Additionally, SmarmOS specifically targets applications with dynamic component graphs, allowing continuous graph reconfiguration. Our approach favors applications with a static internal structure, exposing as many of their characteristics at early stages. Extending application behavior with dynamic aspects will be studied in future work.

Other middleware solutions simplify the development of applications on SSNs by increasing the abstraction level. However, they do not address infrastructure optimizations. For example, the LooCI middleware [23] implements an easy-to-use component composition model, based on event publishing and subscription. This middleware allows for runtime re-configuration and introspection. Such middleware solutions mostly match application needs against sensors capabilities at runtime; they do not attempt to anticipate (mis)matches at earlier phases in application lifecycle to improve their reliability.

Adaptive algorithms: Another body of work concerns adaptive algorithms for sensor networks. This is a highly dynamic approach based on observing application needs and the associated communication patterns, and adapting the network infrastructure to optimize performance. The Adaptive Multi-Criteria Routing (AMCR) algorithm [24] is a key instance of this approach. Its authors argue that most of the WSN routing protocols are designed for a specific application class. AMCR is a generic routing protocol able to adapt to traffic patterns recognized by observing running applications. More specialized routing algorithms, such as the comb-needle routing structure [14], also integrate adaptive behavior based on monitoring an application at runtime and estimating the frequencies of sensor events and application queries. The above-mentioned adaptive approaches are more ambitious than ours, in that their aim is to automatically optimize the sensor infrastructure, without imposing applications to make their needs explicit. However, networks supplying such an advanced self-tuning would need to be extensively tested to ensure the robustness of the adaptive optimizations. Our approach is more predictable in that it exposes sensor-network dimensions of applications such that their execution follows statically defined parameters. Nevertheless, we could re-use the work on adaptive algorithms by leveraging our sensor-network application declarations to perform their optimizations statically, instead of dynamically. As a result, our static approach would incur no runtime overhead.

VII. CONCLUSION

This paper has introduced an approach to developing large-scale orchestrating applications that revolves around a declaration language covering the key sensor-network dimensions. This declaration language is described through a case study of a parking management system. We have shown that our sensor-network declarations can be leveraged across the main stages of an application lifecycle to match the application requirements to a target infrastructure. We have discussed a set of key infrastructure concerns identified in the literature on sensor networks and we have shown how

our approach could take these works further by leveraging information from our declarations.

REFERENCES

[1] Libelium, "Smart City project in Santander to monitor Parking Free Slots," March 2016. [Online]. Available: http://www.libelium.com/smart_santander_parking_smart_city.

[2] Sigfox, "Sigfox," March 2016. [Online]. Available: http://www.sigfox.com.

[3] Worldsensing SL, "Worldsensing and SIGFOX announce the world's largest Intelligent Parking deployment with Micronet, the SIGFOX Network Operator for Russia," March 2016. [Online]. Available: http://www.worldsensing.com/news-press/press-release-worldsensing-and-sigfox-announce-the-worlds-largest-intelligent-parking-deployment-with-micronet-the-sigfox-network-operator-for-russia.html.

[4] B. Raman and K. Chebrolu, "Censor Networks: A Critique of "Sensor Networks" from a Systems Perspective," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 75–78, Jul. 2008.

[5] M. Kabáč, N. Volanschi, and C. Consel, "An Evaluation of the DiaSuite Toolset by Professional Developers: Learning Cost and Usability," in *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU'15)*. ACM, 2015, pp. 9–16.

[6] M. Kabáč and C. Consel, "Orchestrating Masses of Sensors: A Design-Driven Development Approach," in *Proceedings of the 2015 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE'15)*. ACM, 2015, pp. 117–120.

[7] ——, "Designing Parallel Data Processing for Large-Scale Sensor Orchestration," in *13th IEEE International Conference on Ubiquitous Intelligence and Computing (UIC'16)*, 2016., in press.

[8] E. Meshkova, J. Riihijärvi, M. Petrova, and P. Mähönen, "A Survey on Resource Discovery Mechanisms, Peer-to-peer and Service Discovery Frameworks," *Comput. Netw.*, vol. 52, no. 11, pp. 2097–2128, Aug. 2008.

[9] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan, "Building Efficient Wireless Sensor Networks with Low-level Naming," in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP '01)*. ACM, 2001, pp. 146–159.

[10] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks," in *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99)*. ACM, 1999, pp. 263–270.

[11] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman, "A Taxonomy of Wireless Micro-Sensor Network Models," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 6, no. 2, pp. 28–36, Apr. 2002.

[12] W. B. Heinzelman, "Application-Specific Protocol Architectures for Wireless Networks," Ph.D. dissertation, Cambridge, MA, USA, 2000.

[13] J. Heidemann, F. Silva, and D. Estrin, "Matching Data Dissemination Algorithms to Application Requirements," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys '03)*. ACM, 2003, pp. 218–229.

[14] X. Liu, Q. Huang, and Y. Zhang, "Balancing Push and Pull for Efficient Information Discovery in Large-Scale Sensor Networks," *IEEE Transactions on Mobile Computing*, vol. 6, no. 3, pp. 241–251, March 2007.

[15] F. Ye, H. Luo, S. Lu, and L. Zhang, "Wireless Sensor Networks," C. S. Raghavendra, K. M. Sivalingam, and T. Znati, Eds. Kluwer Academic Publishers, 2004, ch. Dissemination Protocols for Large Sensor Networks, pp. 109–128.

[16] J. Mercadal, Q. Enard, C. Consel, and N. Loriant, "A Domain-specific Approach to Architecturing Error Handling in Pervasive Computing," in *Proceedings of the International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA '10)*. ACM, 2010, pp. 47–61.

[17] S. Gatti, E. Balland, and C. Consel, "A Step-wise Approach for Integrating QoS Throughout Software Development," in *Proceedings of the 14th International Conference on Fundamental Approaches to Software Engineering: Part of the Joint European Conferences on Theory and Practice of Software (FASE'11/ETAPS'11)*, 2011, pp. 217–231.

[18] S. Madria, V. Kumar, and R. Dalvi, "Sensor Cloud: A Cloud of Virtual Sensors," *IEEE Software*, vol. 31, no. 2, pp. 70–77, 2014.

[19] F. C. Delicato, P. F. Pires, L. Pirmez, and L. F. Carmo, "A Service Approach for Architecting Application Independent Wireless Sensor Networks," *Cluster Computing*, vol. 8, no. 2-3, pp. 211–221, Jul. 2005.

[20] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis, "TiNA: A Scheme for Temporal Coherency-Aware in-Network Aggregation," in *Proceedings of the 3rd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDe '03)*. ACM, 2003, pp. 69–76.

[21] C. M. D. Farias, W. Li, F. C. Delicato, L. Pirmez, A. Y. Zomaya, P. F. Pires, and J. N. D. Souza, "A Systematic Review of Shared Sensor Networks," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 51:1–51:50, Feb. 2016.

[22] E. A. Lee, B. Hartmann, J. Kubiatowicz, T. S. Rosing, J. Wawrzynek, D. Wessel, J. Rabaey, K. Pister, A. Sangiovanni-Vincentelli, S. A. Seshia, D. Blaauw, P. Dutta, K. Fu, C. Guestrin, B. Taskar, R. Jafari, D. Jones, V. Kumar, R. Mangharam, G. J. Pappas, R. M. Murray, and A. Rowe, "The Swarm at the Edge of the Cloud," *IEEE Design Test*, vol. 31, no. 3, pp. 8–20, June 2014.

[23] D. Hughes, K. Thoelen, W. Horré, N. Matthys, J. D. Cid, S. Michiels, C. Huygens, and W. Joosen, "LooCI: A Loosely-coupled Component Infrastructure for Networked Embedded Systems," in *Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia (MoMM '09)*. ACM, 2009, pp. 195–203.

[24] R. Eltarras and M. Eltoweissy, "Adaptive Multi-Criteria Routing for Shared Sensor-Actuator Networks," in *Global Telecommunications Conference (GLOBECOM'10)*, Dec 2010, pp. 1–6.