Revisiting Service-oriented Architecture for the IoT: A Middleware Perspective

Valérie Issarny, Georgios Bouloukakis, Nikolaos Georgantas, Benjamin Billet

Revisiting Service-oriented Architecture for the IoT: A Middleware Perspective

Valérie Issarny, Georgios Bouloukakis, Nikolaos Georgantas, Benjamin Billet*

MiMove Team, Inria Paris, France.
firstname.lastname@inria.fr

Abstract. By bridging the physical and the virtual worlds, the Internet of Things (IoT) impacts a multitude of application domains, among which smart cities, smart factories, resource management, intelligent transportation, health and well-being to name a few. However, leveraging the IoT within software applications raises tremendous challenges from the networking up to the application layers, in particular due to the ultra-large scale, the extreme heterogeneity and the dynamics of the IoT. This paper more specifically explores how the service-oriented architecture paradigm may be revisited to address challenges posed by the IoT for the development of distributed applications. Drawing from our past and ongoing work within the MiMove team at Inria Paris, the paper discusses the evolution of the supporting middleware solutions spanning the introduction of: probabilistic protocols to face scale, cross-paradigm interactions to face heterogeneity, and streaming-based interactions to support the inherent sensing functionality brought in by the IoT.

Keywords: Internet of things, Interoperability, Middleware, Service-oriented Architecture, Scalability

1 Introduction

The Internet of Things promises the easy integration of the physical world into computer-based systems. In effect, real-world objects become connected to the virtual world, which allows for the remote sensing of as well as the remote acting upon the physical world by computing systems. Improved efficiency and accuracy are expected from this paradigm shift. However, although the vision emerged about 2 decades ago, enacting IoT based systems is still raising tremendous challenges for the supporting infrastructure from the networking up to the programming abstractions. Key challenges relate to ([26]): *scale* as we are dealing with systems that may have to coordinate millions of devices; *deep heterogeneity* since the IoT brings together sensor and actuator networks with cloud-based systems and thus the very small and the very large; *high dynamics* in relation

* The work is supported by the Inria Project Lab CityLab (citylab.inria.fr) and the EU-funded H2020 projects CHOReVOLUTION (chorevolution.eu) and Fiesta-IoT (fiesta-iot.eu).

with the unknown topology of the network and further presence of mobile things as well as uncertainty about the features of the networked things.

The challenges that the IoT is raising in the development of computing systems along with perspectives on how to address them have been the focus of numerous papers over the last decade, such as in: [3, 26, 12, 23]. Among the software architecture paradigms envisioned for IoT-based systems, the literature suggests that service-orientation is promising due to its inherent support for interoperability and composability [13]. A large number of Service-oriented Middleware (SOM) platforms have then been proposed for the IoT, which subsumes revisiting the core elements of the service-oriented architecture paradigm starting with the service abstraction itself.

Towards building a SOM platform for the IoT, the starting point is to abstract Things or their measurements as services [8, 9, 7, 1, 19, 22]. Compared to the classical *Business services*, *Thing-based services* must encompass highly heterogeneous software entities among which resource-constrained ones [2]. An early attempt in that direction is illustrated by the SenseWrap middleware, which features virtual sensors that deal with the transparent discovery of the supporting resources using ZeroConf protocols [19]. The discovery of resource-constrained resources is also the main focus of Hydra [8], *aka* LinkSmart (`https://linksmart.eu`). Hydra further provides interoperability at a semantic level by leveraging semantic Web services technologies for the description of the capabilities of thing-based services. SOCRADES is another one of the early IoT-based SOMs [13], which aims at easing the integration of physical devices into existing enterprise information systems. SOCRADES builds upon the DPWS (Devices Profile for Web Services) standard so that physical Things may expose their resources and may communicate through the Internet as standardized Web services. *EQoSystem* (Emergent QoS System) [20] goes one step further by dealing with the quality of the services delivered by resource-constrained Things. EQoSystem monitors the resources of the underlying devices and triggers resource management strategies (e.g., adapting the service workflow) at runtime for providing acceptable QoS. Similarly in [18], authors investigate a heuristic task allocation algorithm, named *SACHSEN*, which constitutes the core component of their SOM. The algorithm aims to distribute WSN applications to sensor nodes by dealing with the applications' performance requirements and the nodes' energy resources.

In addition to coping with the abstraction of resource-constrained Things as services, the access to the related services also requires special care in IoT-based SOA. We identify two approaches to access a resource-constrained node (typically a sensor or actuator): either *i)* using a proxy/gateway; or *ii)* deploying the middleware component on the sensor/actuator node itself. The former approach was initially favored. However, with the technological evolution of sensor nodes and of SOM, the latter approach is now deserving much attention. Indeed, it provides greater flexibility for managing the physical network infrastructure. The authors in [17] undertake this approach by deploying SOAP-based Web services (DPWS) directly on the nodes without using gateways. Nevertheless, deploying

the middleware component directly on the device might cause several issues, such as message delays, limited supported interactions, limited computational capacity, high energy consumption, etc. Taking into account these problems, the authors in [25] leverage the lightweight CoAP protocol (Constrained Application Protocol, `http://coap.technology/`) on sensor devices and evaluate the trade-off between response times and delivery success rates. Despite the fact that CoAP supports extremely low-resource interactions, it is more suitable for synchronous interactions. Several other protocols have been developed to address the above issues, along with standardization efforts that will guarantee interoperability. The authors in [10] compare the most promising IoT protocols: DPWS for large-scale enterprise deployments, CoAP for lightweight interactions, and MQTT for high reliability. Therefore, it is essential to combine one or more protocols in a WSN application to better exploit the physical network infrastructure.

As briefly outlined above, revisiting SOA and the supporting SOM for the IoT has undergone various steps, as a direct consequence of the technological evolution of the IoT. Obviously, the fact that a growing a number of application domains sees the benefits of leveraging the IoT has also encouraged the development of the enabling hardware and software technologies. This is for instance illustrated by the large literature on middleware solutions for the IoT, as surveyed in [23]. The present paper focuses on the complementary SOM solutions that we have been developing within the MiMove team at Inria Paris to face the scale, dynamics and heterogeneity of the IoT with a special emphasis on enabling resource-constrained Things to become first-class service providers. The next section then outlines our vision of a Thing-based SOA, and is followed by an overview of the supporting SOM solutions in Sections 3 to 5. Section 6 concludes with our perspective for future work.

2 Thing-based SOA

Traditional SOA involves three main actors that interact directly with one another: a Service Provider, a Service Consumer, and a Registry for services. Any service-oriented middleware adopting this architecture supports three core functionalities: *Discovery*, *Composition* of, and *Access* to services. More specifically, *Discovery* is used to publish (register) services in registries that hold service metadata and to look up services that can satisfy a specific request. *Composition* of services is used when discovered services are unable to individually fulfill the request. In such case, existing services are combined to provide a new convenient functionality. The composed services can further be used for more complex compositions. Finally, *Access* enables interaction with the discovered services. This basic SOA architecture is shown in Figure 1.

The IoT brings new requirements and calls for substantially different approaches to the above traditional SOA. Regarding discovery, the principal new challenge is scale when having to deal with millions of Things that produce data of interest, typically sensors that provide real-world measurements. In traditional SOA, even if millions of services are registered, one (or few for backup) is finally

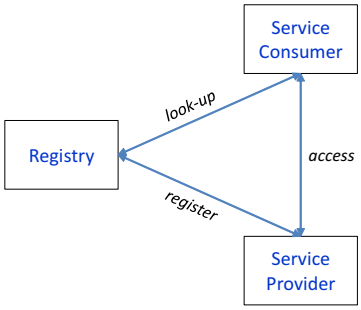


Fig. 1: Traditional Service-Oriented Architecture (SOA).

selected to fulfill a specific request. On the other hand, discovery in the IoT means selecting from a potentially very big number of Things a sufficient subset that will provide, in combination, quality data for a pending query, while limiting unnecessary redundancy for the sake of the scarce resources. Furthermore, a considerable portion of the networked Things that connect with the physical world are endowed with the ability to change their location either autonomously or, for instance, with human involvement (e.g., mobile phones, vehicles, etc.). These mobile Things are now within everyone's reach. For instance, all mobile phones nowadays host at least two sensors, a camera and a microphone. As of 2011, there were 5.3 billion phones users, of whom more than 1 billion owned a smartphone[1] with additional sensors, such as gyroscopes and barometers. Another example is the increasing integration of sensors and actuators in vehicles. We look into the problem of scalable discovery of Things – both fixed and mobile – in Section 3 and report on our related research results.

Concerning composition, the key challenge is resource saving, especially when it comes to dealing with continuous complex processing of data streams produced by numerous Things that are resource constrained. While, in traditional SOA, a composite service typically involves an exchange (direct or indirect) of a few discrete messages between the constituent services, in IoT data streaming, big volumes of data need to be collected from sensors, processed, composed and finally stored or delivered to actuators. Even if relying on the cloud is a widely adopted solution to this challenge, this incurs high communication and energy cost for Things and networks. We discuss this issue along with our in-network solution approach in Section 4.

Finally, access is essential for any IoT deployment, whether there is direct communication among Things or through the cloud. The hard challenge here is heterogeneity, which is particularly acute in the highly fragmented IoT world and concerns all hardware and software aspects of Things. In traditional SOA, standardization has been particularly effective, with WS-* and REST web services being the two dominant technologies. Regarding the same aspect in the

[1] US Strategy Analytics: www.strategyanalytics.com.

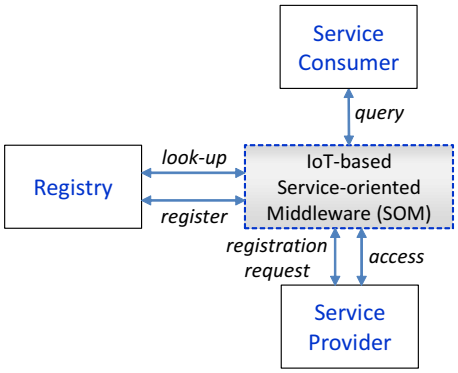


Fig. 2: Thing-based SOA.

IoT, i.e., public service description and middleware-level service access, where we assume Things sufficiently powerful to support IP protocol stacks, there is much bigger diversity. Message-, event- and data-based interaction styles are all widely used in the IoT with various protocol realizations. We analyze this issue and provide an overview of our current related research in Section 5.

3 Discovery in the Ultra-large Scale IoT

Real-world measurements in the IoT require a large number of Things, since it is unlikely for a single or even a few Things to be sufficient. Applying the proposed traditional approaches of SOA to discover Things (i.e, discover all the appropriate devices that are reachable), will return a large set of accessible Things, many of which provide redundant functionalities. Moreover, mobility in the IoT constitutes an additional challenge regarding the discovery of sensors embedded in mobile devices.

More specifically, the application of SOA to the IoT results in some apparent problems. On the one hand, all the tasks in SOA revolve around some business logic that can be satisfied by one or several services. On the other hand, in the IoT, all the tasks and interactions revolve around what we refer to as a Thing-based query that senses/actuates some real world phenomenon. An example of a Thing-based query would be "What is the air pollution level on highways in Paris?". Thus, with such queries it is unlikely to have only one or just a few services that can provide accurate answers to represent a real-world feature. Hence, expecting the service consumer to interact with the numerous relevant service providers individually to access their services and acquire their measurements, then know how to treat each and every value (with different possible formats, types, units, etc.), in addition to the aggregation logic to apply, requires high communication and computation capabilities that the consumer will most likely not possess.

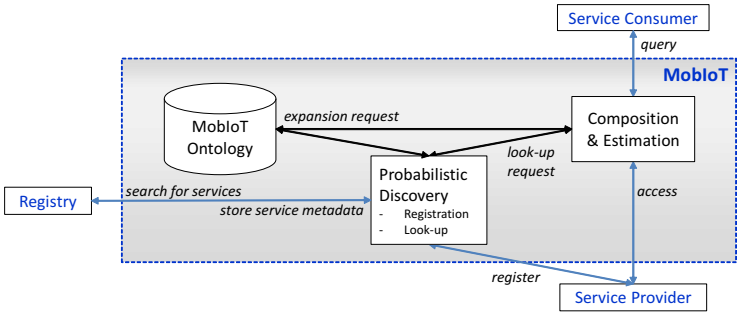


Fig. 3: MobIoT architecture.

As an alternative, our approach revisits the SOA and renders most interactions and heavy computations transparent to the consumer, who is only expected to know the sought after measurements. Figure 2 depicts the Thing-based SOA that is to be contrasted with the traditional SOA of Figure 1. This has further led us to introduce the supporting *MobIoT* service-oriented middleware [16]. As depicted in Figure 3, MobIoT supports the core functionalities of SOA (discovery, composition and access), while enabling Thing-based queries. In particular, MobIoT features a novel probabilistic discovery protocol. In MobIoT, the Discovery component wraps *Registration* and *Look-up* functionalities as follows:

- *Probabilistic Registration:* the registration of a provider's service is probabilistic. The goal is to allow only a subset of willing providers to register their services, depending on whether the already registered ones are sufficient. More precisely, in MobIoT, the *Registration component* generates the decision to allow or prevent a Thing from registering its services. The component estimates whether or not the mobility path to be followed by this Thing can be covered by other, already registered, mobile Things with similar sensing/actuating services. To that end, the component computes the probability that any of the latter Things be present at each of the future locations on the path of the former Thing when the former Thing crosses them. Then, the component compares the resulting probability value to a required sensing coverage. We consider that the coverage requirement (threshold) depends on the sensor and can be specified in its metadata. Only if the resulting probability is lower than the threshold, the Thing registers its service. We use the *Truncated Lévy Walk* mobility model [24] to estimate the mobility of registered Things and compute the probabilities above. The Registration component can use any other mobility model as long as the corresponding mathematical formulas to compute the probabilities are provided. As shown in [15], our registration solution successfully limits the registration of redundant services.
- *Probabilistic Look-up:* the look-up is also probabilistic and returns only a subset of sensing services based on the total area coverage they can pro-

vide [14]. We adopt the same logic as in intrusion detection solutions [27], where the spatial distribution of sensors has a major effect on the performance of the sensing system. Based on those solutions, when measuring a feature over some area (e.g., Air quality level), we sample sensors from a Uniform distribution in space, so that sensors from all over that area have the same likelihood of being selected. However, when the concept of interest is at a specific point in space, a better distribution would be Normal, as it selects more sensors around that point and less as we move farther. More specifically, in MobIoT, the *Look-up component* is in charge of returning a subset of services to access that can satisfy the Thing-based query. Based on the requested measurement and the location of interest, the component determines the most adequate probability distribution and the number of needed services. This number is computed based on the coverage requirement, expressed as a percentage of the area of interest to be sensed/acted upon by the selected subset. The result is then forwarded to the Registry to determine the actual Things to sample.

4 Composition in the Dynamic Resource-constrained IoT

A major feature of a SOA is supporting the composition of the operations of existing services in order to create composite services [21]. Contracts describe the inputs and the outputs of service operations and enable the specification of a workflow of service invocations that represents the logic of a new more complex operation. In practice, composite services execute in a centralized or a distributed fashion. The former approach, called *service orchestration*, introduces an orchestrator to manage the invocations of the underlying services. The latter approach, called *service choreography*, leverages negotiation and routing mechanisms to let the service providers manage the composite services autonomously.

Applying a similar approach in the IoT is not a trivial procedure. We are, in particular, interested in the composition of Things that collaboratively produce, process and consume IoT data streams. Our emphasis is on in-network continuous processing of sensed data streams, as opposed to delegating all of the computation to the cloud, aiming to reduce the energy cost of communication over computation incurred by the massive scale of the IoT. Additionally, the IoT is characterized by a network topology that may be unknown and highly dynamic, due to the mobility of Things or their short life span. As a consequence, services required by an IoT application may suddenly become unavailable, because the host ran out of battery or just changed its location abruptly. To deal with all the above issues, the traditional composition of SOA must be extended in order to infer which service providers have to be used for executing the services, according to a set of scenario-dependent properties (e.g., throughput, energy consumption). This problem is a variation of the *task mapping problem*, where a set of communicating tasks with several properties (constraints, resource consumption, etc.) must be mapped to a set of connected nodes given their characteristics (location, hardware capabilities, etc.).

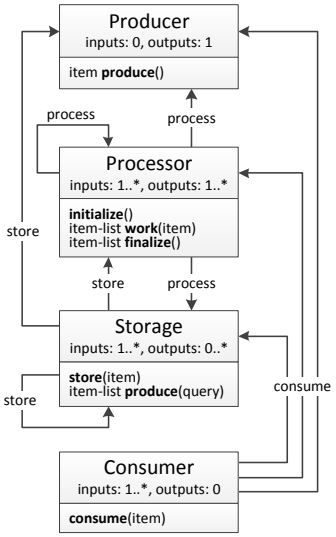
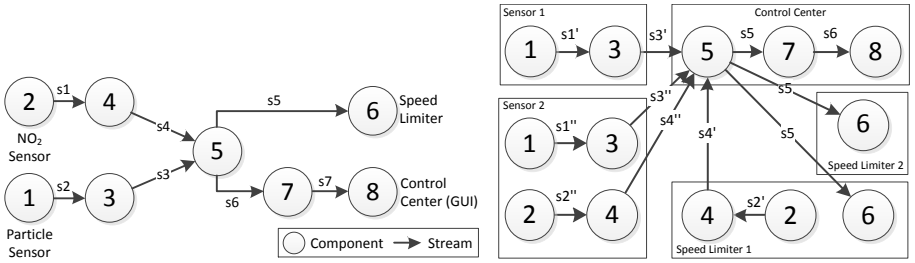


Fig. 4: The four-roles model.

Both centralized and distributed approaches have been studied to solve the above problem. The centralized approach computes and executes an allocation plan on a single machine (similar to service orchestration). The distributed approach lets the nodes compute parts of the allocation plan based on the knowledge they have about their peers (similar to service choreography). In MiMove, we have studied this problem and have introduced *Dioptase*, a distributed data streaming middleware for the Web of Things [4]. Dioptase makes it possible to: *i)* integrate the Things with today's Web by exposing sensors and actuators as Web services, *ii)* manage physical data as streams, and *iii)* use any Thing as a generic pool of resources that can process streams by running tasks that are provided by developers over time.

More concretely, we introduce in [5] a dedicated centralized solver (Dioptase component) to allocate tasks on resource-constrained Things that produce data as streams. Depending on the available resources, each Thing may play one (or more in combination) of the identified four high-level roles (Figure 4): *i)* a production role where the Thing presents sensor data as streams, *ii)* a processing role where the Thing continuously processes streams, *iii)* a consumption role where the Thing acquires streams and drives actuators, and *iv)* a storage role where the Thing saves data extracted from streams (in its memory, or persistently).

Subsequently, a *Dioptase mashup* is composed of distributed components, called *atomic components*, derived from the above roles. These components interact (are connected) by continuously exchanging data as streams. Each component defines *input ports* for the consumption of streams, depending on the component type, and *output ports* where new stream items are produced. Pro-



(a) Logical mashup graph. (b) Physical mashup graph.

①	Producer	Reads the PM_{10} sensor every x seconds.
②	Producer	Reads the NO_2 sensor every x seconds.
③	Processor	Every y minutes, computes the average of the PM_{10} measurements between min (PM_{10}) and max (PM_{10}).
④	Processor	Every y minutes, computes the average of the NO_2 measurements between min (NO_2) and max (NO_2).
⑤	Processor	Aggregates geographically the values of ③ and ④. Produces two streams: *i)* a stream of events *decrease-speed* or *increase-speed*, depending on the aggregated results ($s5$) and *ii)* a stream containing the aggregated results ($s6$).
⑥	Consumer	Increases or decreases the displayed speed limit, according to the events received from ⑤.
⑦	Storage	Stores the results of ⑤.
⑥	Consumer	Asks for data stored in ⑦ and presents it to the application.

Fig. 5: Logical and physical mashup graphs for air control pollution.

vided the data types specified for the input and output streams match, any output port can be connected to any input port through a one-to-one connection. We represent the above components as services of SOA. The ports provided by a service define its *interface*, while the *contract* specifies the schemas of *i)* the streams readable by each input port and *ii)* the streams produced by each output port. Given a contract, the service consumer can reason about the operations of the service and the streams that each operation processes and produces.

The *mashup* can then be easily described as an acyclic directed graph where the nodes are producers (sources), processors, consumers (sinks) and storages, and the edges of the graph, are streams that link services together through the input and output ports. We call this graph a *logical mashup graph* because it describes the tasks that the network has to perform. This graph is provided by the developer either directly or expressed as a query that is translated into a mashup graph. As an illustration, Figure 5a presents an example of a simple mashup that analyzes air pollution based on the level of nitrogen dioxide (NO_2) and particulate matter (PM_{10}), in order to control the digital speed limiters located along city highways. In this mashup, two producers (① and ②) read the PM_{10} and NO_2 values from available sensors. Those data are acquired by

two processors (③ and ④) that aggregate the values on a 10 minute basis (time window). Processors ③ and ④ send these aggregated results to the processor ⑤ that will produce an event stream for the speed limiters ⑥. At the same time, the measurements are saved by a storage ⑦ and consumed by the air pollution control application ⑧, which presents historical values or alerts to an administrator.

Subsequently, the *logical mashup graph* is executed through its conversion into a *physical mashup graph*, as depicted in Figure 5b. The execution of a logical graph is done through: *i)* instantiating the services (i.e., maps a service onto a host device) and *ii)* connecting their ports according to the graph edges. Regarding the latter, the data exchanged between two services is pull-based (where a consumer requests a producer to send the data stream), and Dioptase connects the services' ports (input or output) that are specified in each contract. After executing the logical graph, the services are hosted on the following devices (Figure 5b): a PM_{10} sensor (Sensor 1), a $PM_{10} + NO_2$ sensor (Sensor 2), a speed limiter embedding an NO_2 sensor (Speed Limiter 1), a speed limiter embedding no sensors (Speed Limiter 2), and the control center computer. According to this physical deployment, an instance of Processor ③ will execute on the same device that hosts Producer ①, thus reducing network traffic. Moreover, processor ⑤, which controls the speed limiters, will run on the control center computer together with the history database ⑦ and the air pollution control application ⑧.

To instantiate the services as illustrated in the example, we use information about Things' locations and available resources. Then, depending on its capabilities, a Thing can be assigned either a single component or an entire subgraph by using the task mapping algorithm that we have proposed in [5]. In brief, we formalize therein the task mapping problem in the specific context of the IoT, which results in a binary programming problem. We provide a heuristic algorithm to solve it and demonstrate experimentally the efficiency, sufficient optimality, and reasonable resource requirement of our solution. Consequently, the mapping can be performed directly within the network, without requiring any centralized infrastructure.

5 Access in the Heterogeneous IoT

The *access* mechanism of traditional SOA enables the interaction between service consumers and service providers. In particular, services interact in a unified way following specific data formats on top of common overlay infrastructures across different system platforms. Web services constitute the dominant technology in SOA, with well known protocols such as SOAP or REST as the overlay infrastructure. The research community and many businesses have adopted these protocols and their standards in order to describe and implement their services (i.e., the supported operations, data formats, etc.). Also regarding the interconnection of these protocols, the existence of standards facilitates the development of frameworks for interoperability.

On the other hand, the (mobile) IoT comprises sensors and actuators that are heterogeneous with different operating (e.g., operating platforms) and hardware (e.g., sensor chip types) characteristics, hosted on diverse Things (e.g., mobile phones, vehicles, clothing, etc.). To support the deployment of such devices, major tech industry actors have introduced their own APIs and protocols, which deal with: *i)* the limited energy resources of Things; *ii)* several data formats found in the IoT; *iii)* specific guarantees regarding response times and data delivery success rates; *iv)* the efficient transfer of small data payloads which are common in the IoT; etc. The resulting APIs and protocols are highly heterogeneous. In particular, protocols differ significantly in terms of interaction styles and data formats. For instance, protocols such as CoAP relying on *client-service* interactions, MQTT based on the *publish-subscribe* interaction paradigm, SemiSpace offering a lightweight shared *tuple space*, or Websockets based on *streaming* interactions, are among the most widely employed ones.

Hence, providing *access* to Things establishes a new challenge with respect to traditional SOA. To deal with this challenge, we have introduced the *eVolution Service Bus (VSB)*. *VSB* is a development and runtime environment dedicated to complex distributed applications. Its objective is to seamlessly interconnect, Things that employ heterogeneous interaction protocols at the middleware level (e.g., DPWS, CoAP, MQTT, Dioptase, etc.). This is based on runtime conversions between such protocols, with respect to their primitives and data type systems, while properly mapping between their semantics. This also includes mapping between the public service interfaces of Things, regarding their operations and data, from the viewpoint of the middleware: the latter means that operations and data are converted based on their middleware-level semantics, while their business semantics remains transparent to the conversion. VSB follows the well-known Enterprise Service Bus (ESB) paradigm [6]. In this paradigm, a *common intermediate bus protocol* is used to facilitate interconnection between multiple heterogeneous middleware protocols: instead of implementing all possible conversions between the protocols, we only need to implement the conversion of each protocol to the common bus protocol, thus considerably reducing the development effort. This conversion is done by a component associated to the Thing in question and its middleware, called a *Binding Component (BC)*, as it enables the interaction between the Thing and the common bus protocol. VSB follows a fully distributed architecture implemented by a number of Binding Components (BCs) that interact among themselves through the VSB common bus protocol.

A view of the VSB architecture is depicted in Figure 6, showing the interconnection of *Sensor 2* and the *Control Center* of the physical mashup in Figure 5b. Within every mashup, there are streams of data exchanged using the Dioptase middleware. However, Sensor 2 publishes data through an MQTT middleware component and the Control Center accepts data through a CoAP component. Thus, using VSB we enable the interconnection of heterogeneous protocols. Particularly, *BC 1* is associated to *Sensor 2*, while *BC 2* is associated to the *Control Center*. Implementation-wise, a BC employs the same (or symmetric, e.g., client vs. server) middleware protocol library as its associated Thing, and all BCs use

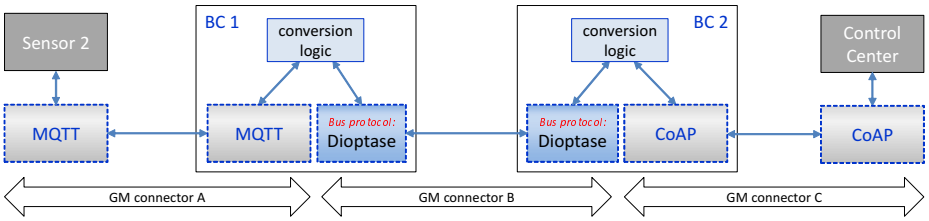


Fig. 6: VSB architecture.

a library implementing the bus protocol, which in our case is the Dioptase middleware. MQTT, CoAP and Dioptase are treated in the same way within the VSB architecture. More specifically, each end-to-end interaction using the same middleware-layer protocol is modeled and abstracted by the *Generic Middleware (GM) connector*, as indicated in Figure 6.

The GM connector abstracts interactions among peer components that employ the same middleware protocol in a unifying fashion for any middleware protocol. We propose an API (application programming interface) for GM and a related interface description, which we call *GIDL (Generic Interface Description Language)*, for application components that (abstractly) employ GM. Concrete middleware protocols and related interface descriptions of application components that employ these middleware protocols can be mapped to GM API and GIDL, respectively. Based on these abstractions, we elaborate a generic architecture for BCs (which we call *Generic BC*), an *Implementations' Pool*, which contains implementations of the GM API for concrete middleware or bus protocols, as well as a related method for *BC Synthesis*.

We provide more details concerning the *GM connector*, *GIDL*, and *BC synthesis* in the following:

- *Generic Middleware (GM) Connector*: based on our experience with middleware protocols/paradigms and their modeling in [11], we introduce a detailed API for GM. The GM API identifies and supports basic interaction styles found in most middleware protocols: *one-way interaction*, *two-way asynchronous interaction*, *two-way synchronous interaction*, and *stream interaction*. It also distinguishes between the two roles involved in an interaction, such as: *provider* and *consumer*. Essentially, the API relies on two main actions: a *post* action for sending a piece of data and a *get* action for receiving a piece of data.
- *Generic Interface Description Language (GIDL)*: by relying on the GM API, we elicit a generic interface description (GIDL) for a Thing that employs a middleware protocol abstracted by GM. *GIDL* enables the definition of operations provided or required by a Thing and that follow the interaction styles and roles identified in the GM API. Besides an operation's type, the names and data types of its parameters are also specified. The description is complemented by the physical address of the Thing.

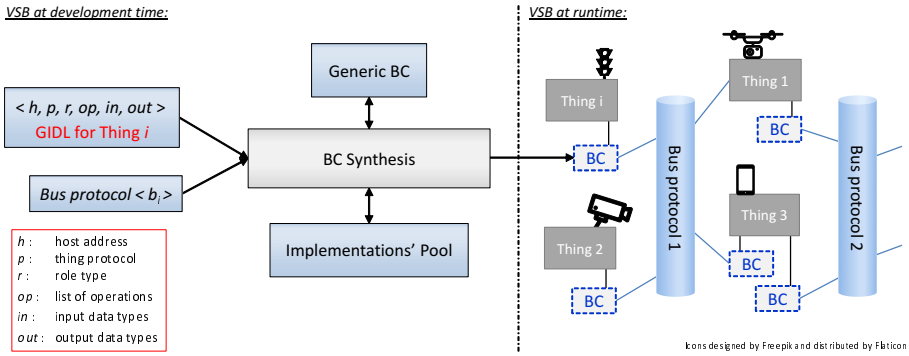


Fig. 7: VSB development and runtime environments.

– *BC Synthesis*: the functioning of *BC Synthesis* for the generation of a BC intended to serve a specific Thing is illustrated in Figure 7, which depicts VSB's development and runtime environments. *BC synthesis* receives as input the GIDL description of the Thing and the information of the bus protocol in use. Based on this input, *BC synthesis* refines the Generic BC in two steps: *(1)* by selecting appropriate GM API implementations from the Implementation's Pool that correspond to the Thing's middleware protocol and bus protocol; and *(2)* by inserting specific information about the Thing's operations and data from its GIDL description. The outcome of *BC synthesis* is a BC that will be deployed to serve the Thing.

VSB enables *access* to heterogeneous Things, while taking into account the current diversity but also the future evolution of IoT protocols. Hence, as shown in Figure 7, a Thing may participate in more than one runtime topologies, which can be readily supported by multiple BCs and buses. Additionally, depending on the constraints found in an application scenario (e.g., devices with limited energy resources), any new protocol can be introduced as the VSB's common bus protocol. Accordingly, BCs are built and deployed as necessary: no BC is needed when a Thing employs the same middleware protocol as the bus protocol. Finally, there is no need for relying on and/or providing a full-fledged ESB platform (e.g., a cloud-based platform), which makes the VSB solution particularly flexible and lightweight.

6 Conclusion

IoT-based SOA holds the promise of easing the development of rich applications integrating the physical with the virtual worlds in a multitude of domains. This paper has presented our perspective on the definition of a supporting Service-oriented Middleware, which primarily revolves around enabling the provision of services by resource-constrained Things, typically sensors and actuators. Chal-

lenges then relate to: dealing with the ultra-large number of Things that are expected to be deployed in most environments, composing the services offered by the Things while coping with their inherent resource limitations and dynamics, and accessing the various networked things despite their high heterogeneity, including in terms of supported communication protocols. We have been studying solutions to these issues, which has led us to revisit the core functions of a SOM: service discovery becomes probabilistic to filter out redundant Things, especially accounting for the possible mobility of the service clients and/or providers; service composition aggregates data streams *within* the network so as to reduce the network load; and service access enables the interconnection of Things that adhere to different interaction styles (spanning client-server, event-based and data sharing communication). While those middleware solutions have led to the development of different SOM instances, we are now studying their integration so as to support the development of application toward smarter cities, in particular in the area of urban pollution monitoring.

References

1. Aberer, K., Hauswirth, M., Salehi, A.: Infrastructure for data processing in large-scale interconnected sensor networks. In: 2007 International Conference on Mobile Data Management. pp. 198–205. IEEE (2007)
2. Athanasopoulos, D., Autili, M., Georgantas, N., Issarny, V., Tivoli, M., Zarras, A.: An Architectural Style for the Development of Choreographies in the Future Internet. Global Journal of Advanced Software Engineering 1(1), 14–28 (Dec 2014), `https://hal.inria.fr/hal-01110502`
3. Atzori, L., Iera, A., Morabito, G.: The internet of things: A survey. Comput. Netw. 54(15), 2787–2805 (Oct 2010)
4. Billet, B., Issarny, V.: Dioptase: a distributed data streaming middleware for the future web of things. Journal of Internet Services and Applications 5(1), 28 (2014)
5. Billet, B., Issarny, V.: From task graphs to concrete actions: a new task mapping algorithm for the future internet of things. In: 2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems. pp. 470–478. IEEE (2014)
6. Chappell, D.A.: Enterprise Service Bus. O'Reilly Media (2004)
7. Corredor, I., Martínez, J.F., Familiar, M.S., López, L.: Knowledge-aware and service-oriented middleware for deploying pervasive services. Journal of Network and Computer Applications 35(2), 562–576 (2012)
8. Eisenhauer, M., Rosengren, P., Antolin, P.: Hydra: A development platform for integrating wireless devices and sensors into ambient intelligence systems. In: The Internet of Things, pp. 367–373. Springer (2010)
9. Fok, C.L., Roman, G.C., Lu, C.: Servilla: a flexible service provisioning middleware for heterogeneous sensor networks. Science of Computer Programming 77(6), 663–684 (2012)
10. Fysarakis, K., Askoxylakis, I., Manifavas, C., Soultatos, O., Papaefstathiou, I., Katos, V.: Which iot protocol? comparing standardized approaches over a common m2m application. IEEE Global Communications Conference (2016)
11. Georgantas, N., Bouloukakis, G., Beauche, S., Issarny, V.: Service-oriented Distributed Applications in the Future Internet: The Case for Interaction Paradigm Interoperability. In: Euro. Conf. on Service-Oriented and Cloud Computing (2013)

12. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (iot): A vision, architectural elements, and future directions. Future Gener. Comput. Syst. 29(7), 1645–1660 (Sep 2013), http://dx.doi.org/10.1016/j.future.2013.01.010
13. Guinard, D., Trifa, V., Karnouskos, S., Spiess, P., Savio, D.: Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. IEEE transactions on Services Computing 3(3), 223–235 (2010)
14. Hachem, S.: Service-Oriented middleware for the large-scale mobile Internet of Things. Ph.D. thesis, Université de Versailles-Saint Quentin en Yvelines (2014)
15. Hachem, S., Pathak, A., Issarny, V.: Probabilistic registration for large-scale mobile participatory sensing. In: Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on. pp. 132–140. IEEE (2013)
16. Hachem, S., Pathak, A., Issarny, V.: Service-oriented middleware for large-scale mobile participatory sensing. Pervasive and Mobile Computing 10, 66–82 (2014)
17. Kyusakov, R., Eliasson, J., Delsing, J., van Deventer, J., Gustafsson, J.: Integration of wireless sensor and actuator nodes with it infrastructure using service-oriented architecture. IEEE Transactions on industrial informatics 9(1), 43–51 (2013)
18. Li, W., Delicato, F.C., Pires, P.F., Lee, Y.C., Zomaya, A.Y., Miceli, C., Pirmez, L.: Efficient allocation of resources in multiple heterogeneous wireless sensor networks. Journal of Parallel and Distributed Computing 74(1), 1775–1788 (2014)
19. Meling, H., et al.: Sensewrap: A service oriented middleware with sensor virtualization and self-configuration. In: Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2009 5th International Conference on. pp. 261–266. IEEE (2009)
20. Newman, P., Kotonya, G.: A resource-aware framework for resource-constrained service-oriented systems. Future Generation Computer Systems 47, 161–175 (2015)
21. Papazoglou, M.P.: Service-oriented computing: Concepts, characteristics and directions. In: Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on. pp. 3–12. IEEE (2003)
22. Perera, C., Jayaraman, P.P., Zaslavsky, A., Christen, P., Georgakopoulos, D.: Mosden: An internet of things middleware for resource constrained mobile devices. In: 2014 47th Hawaii International Conference on System Sciences. pp. 1053–1062. IEEE (2014)
23. Razzaque, M.A., Milojevic-Jevric, M., Palade, A., Clarke, S.: Middleware for internet of things: A survey. IEEE Internet of Things Journal 3(1), 70–95 (Feb 2016)
24. Rhee, I., Shin, M., Hong, S., Lee, K., Kim, S.J., Chong, S.: On the levy-walk nature of human mobility. IEEE/ACM transactions on networking (TON) 19(3), 630–643 (2011)
25. Sheng, Z., Wang, H., Yin, C., Hu, X., Yang, S., Leung, V.C.: Lightweight management of resource-constrained sensor devices in internet of things. IEEE internet of things journal 2(5), 402–411 (2015)
26. Teixeira, T., Hachem, S., Issarny, V., Georgantas, N.: Service oriented middleware for the internet of things: A perspective. In: Proceedings of the 4th European Conference on Towards a Service-based Internet. pp. 220–229. ServiceWave'11, Springer-Verlag, Berlin, Heidelberg (2011)
27. Wang, Y., Fu, W., Agrawal, D.P.: Gaussian versus uniform distribution for intrusion detection in wireless sensor networks. IEEE Transactions on Parallel and Distributed systems 24(2), 342–355 (2013)