Analysis and Classification of Service Interactions for the Scalability of the Internet of Things

Damian Arellanes and Kung-Kiu Lau
School of Computer Science
The University of Manchester
Manchester M13 9PL, United Kingdom
{damian.arellanesmolina, kung-kiu.lau}@manchester.ac.uk

***Abstract*—Scalability is an important concern for Internet of Things (IoT) applications since the amount of service interactions may become overwhelming, due to the huge number of interconnected nodes. In this paper, we present an IoT scenario for real-time Electrocardiogram (ECG) monitoring, in order to analyze how well different kinds of service interactions can fulfill the scalability requirements of IoT applications.**

***Index Terms*—Internet of Things (IoT), service interactions, scalability, Internet of Services (IoS)**

I. Introduction

The Internet of Things (IoT) promises a new era in which not only people interact through Internet, but so do things. Currently, the number of connected devices worldwide is about 17 billion, and it is estimated that this number will grow by a factor of 1.82 in the next three years [1]. For this reason, scalability in terms of the size of IoT applications, rather than vertical or horizontal scalability [2], is an important concern.

For this kind of scalability, four crucial desiderata has been identified: explicit control flow [3], separation between control and computation [4], decentralization [5] and location transparency [6]. In this paper, we analyze how well different kinds of service interactions can fulfill these scalability requirements.

Service interactions play a central role in the Internet of Services (IoS) [7] which will be a key enabler of the IoT goals. IoT services interact in different ways to achieve a common goal in a specific application. Despite an increasing number of proposed network protocols for IoT, there is a lack of understanding about which service interactions best fulfill the scalability requirements of IoT applications.

The rest of the paper is structured as follows. Sect. II presents an IoT scenario for real-time Electrocardiogram (ECG) monitoring. Sect. III describes our classification of service interactions. Sect. IV presents the results of our analysis. Sect. V presents a discussion of our results. Finally, Sect. VI presents the conclusions and the future work.

II. IoT Scenario: Electrocardiogram Monitoring Network (EMoNet)

This section introduces a running example for the rest of the paper. The example is an IoT scenario for real-time Electrocardiogram (ECG) monitoring: Electrocardiogram Monitoring Network (EMoNet). EMoNet is a network deployed in a smart city, consisting of patients with cardiac diseases, plenty of ambulances moving around the city, patients' smartwatches and wearable ECG sensors. Fig. 1 depicts the workflow of EMoNet which corresponds to a timing task triggered every 3 minutes for a particular patient. It basically consists of pulling and analyzing ECG data, and requesting the nearest ambulance if the patient has heart attack signs.

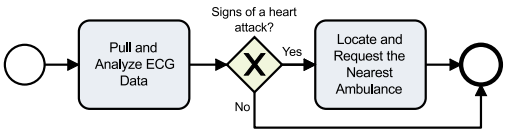

Fig. 1. EMoNet workflow.

The EMoNet workflow involves four independent IoT nodes shown in Fig. 2: a wearable ECG sensor installed on a patient's chest, the patient's smartwatch, a healthcare cloud and an ambulance.

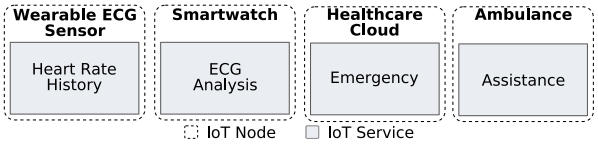

Fig. 2. Nodes and services involved in the EMoNet workflow.

The wearable ECG sensor provides the *Heart Rate History* service as an interface for the records of the electrical activity of a patient's heart. The smartwatch provides the *ECG Analysis* service that determines if a patient is showing signs of a heart attack. A healthcare cloud provides the *Emergency* service to find the nearest ambulance and request it immediately. Ambulances provide the *Assistance* service to attend to those patients in need on-site. For simplicity, we assume that these services dispatch many requests concurrently. In the next section, we will describe different ways to realize the EMoNet workflow using these services.

III. Service Interaction Schemas

IoT services provide low-level functionality implemented in nodes [8]. Resource-constrained nodes (e.g., a pulse sensor) provide fine-grained services for basic functionality (e.g., fetching sensor data). Non resource-constrained nodes (e.g., a smart TV) may offer coarse-grained services in addition.

978-1-5386-7244-0/18/$31.00 ©2018 IEEE
DOI 10.1109/ICIOT.2018.00018

IoT services interact via a network in order to realize complex functionality. Services can interact by message passing, event exchanges, or any combination thereof. In order to determine what kind of interactions best fulfills the scalability requirements of IoT, we have classified service interactions into four schemas: (i) direct service interactions, (ii) indirect service interactions, (iii) exogenous service interactions and (iv) event-driven service interactions.

Schemas (i), (ii) and (iii) are based on message-passing, where there are two roles: service sender and service receiver. A service sender accesses functionality offered by a service receiver, by passing a message (expressing control) via the network. Schema (iv) is based on events so a service registers itself with events that will be produced by another service(s). In this section, we describe these four schemas in more detail.

Microservice Architecture [9] has gained considerable attention in the last few years, and is becoming increasingly important and popular for the development of IoT applications [10]. Every Microservice Architecture is a Service-Oriented Architecture (SOA), but not the other way round [11]. Hence, the service interaction schemas presented in this section can be used interchangeably in both Microservices and traditional SOA services.

A. Direct Service Interactions

The direct service interaction schema consist of sending a message (e.g., a XML-based document or a JSON-based document) from a sender to a receiver with no mediator between them [12], [13]. The sender interacts with the receiver using Remote Procedure Calls (RPC) [14] or REST API calls over HTTP [15]. RPC is akin to method invocations in traditional Object-Oriented programming languages, the main difference being that the invoked procedures may reside at different network addresses. REST does not require to know procedure names in advance, but only the location of external resources that can be manipulated using HTTP methods. Direct interactions are typically done using the request-response pattern [16].

Fig. 3 illustrates direct interactions for the EMoNet workflow. *ECG Analysis* triggers the control flow periodically by passing control to *Heart Rate History* so as to get the last sensor reading. Then, *Heart Rate History* returns the control to *ECG Analysis*. If there are signs of a heart attack, *ECG Analysis* passes control to the *Emergency* service which forwards control to the *Assistance* service of the nearest ambulance. Control is returned to *ECG Analysis*, after passing through the *Emergency* service. Fig. 3 shows that data flow follows control flow, and control and data are always originated in service computation.

Although they look old-fashioned, direct interactions are being used in emerging technologies (including IoT). For example, a Microservice choreography [17] describes direct interactions which are typically done using RESTful APIs [18]. REST has also been fostered by the Web of Things [8] for direct interactions among IoT services via the Web. Moreover, recent server-less programming frameworks for IoT [19] enable Java RPC for direct service interactions.

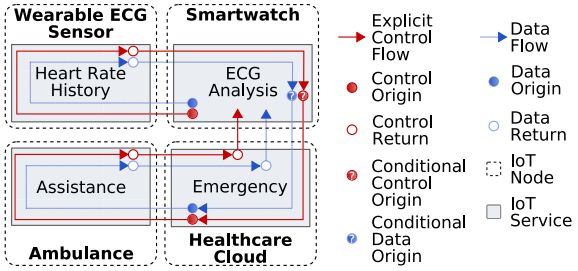

Fig. 3. Direct service interactions for the EMoNet workflow.

B. Indirect Service Interactions

The indirect interaction schema consists of using a service bus to broker sender requests, locate an appropriate receiver, transmit requests, and send responses back to senders. Since it passes messages between senders and receivers, a service bus can be thought of as a universal connector that provides a level of indirection between services [20], [21].

Fig. 4 illustrates indirect interactions for the EMoNet workflow, where *ECG Analysis* triggers control flow periodically. EMoNet services register their interfaces with a service bus that forwards control (and data) originated by *ECG Analysis* and *Emergency*, and sends back control (and data) from *Heart Rate History*, *Assistance* and *Emergency*, respectively. A glance at Fig. 4, reveals that even though a service bus provides indirection between senders and receivers, control and data are originated in service computation, and data follows control.

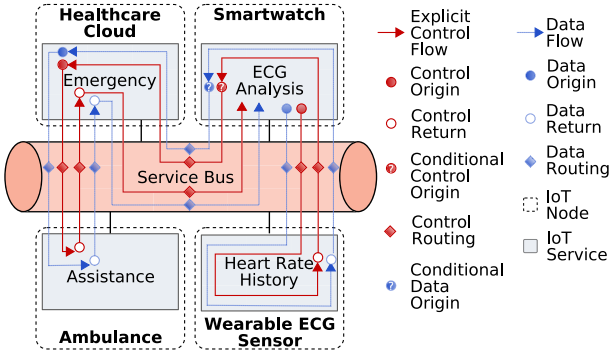

Fig. 4. Indirect service interactions for the EMoNet workflow.

Although the Enterprise Service Bus (ESB) [20] has been used for over a decade for enterprise SOA applications, the Microservice Architecture community has recently expressed their interest of using a lighter bus (known as Gateway) for indirect Microservice interactions [22], [23]. An IoT application can use a Gateway, an ESB or both [24].

C. Event-Driven Service Interactions

The event-driven interaction schema is based on the publish-subscribe pattern [16] so there are two roles: producer (i.e., service sender) and consumer (i.e., service receiver). Producers trigger events (perhaps carrying data) which are then stored in a queue. Consumers can subscribe to the events they are interested in, retrieve those events from the queue and react

accordingly. As events are dequeued in FIFO mode, there is no guarantee that responses from consumers are delivered to producers, so event-driven interactions usually follow the principle *fire and forget* [25], [26], [27].

Event-driven interactions can be done with or without a service bus. *P2P event-driven interactions* enable every service to be responsible of its own queue, so events are exchanged with no mediator. ZeroMQ is the most popular library to realize this interaction schema.[1] Fig. 5(a) shows P2P event-driven interactions for the EMoNet workflow.

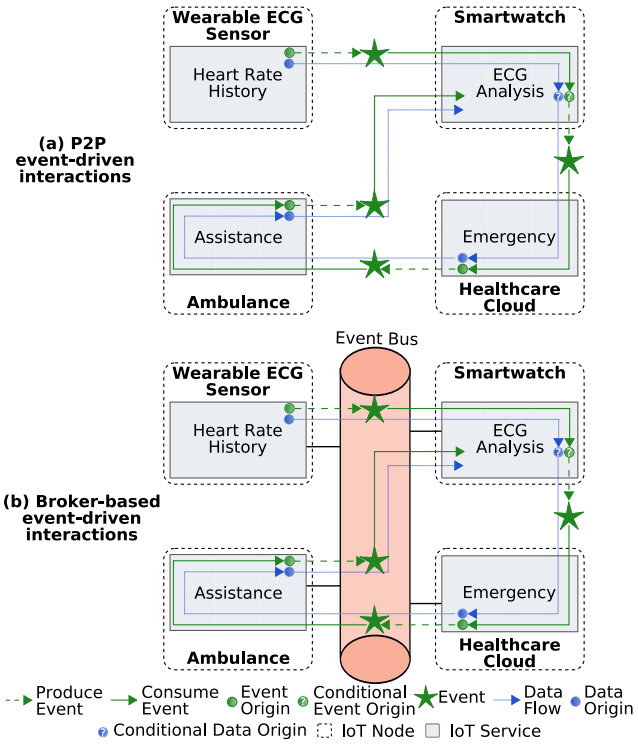

Fig. 5. Event-driven service interactions for the EMoNet workflow.

ECG Analysis periodically gets the last sensor readings by consuming events produced by *Heart Rate History*. If it detects a heart attack, *ECG Analysis* announces an emergency situation by producing an event for *Emergency*. After determining the nearest ambulance, *Emergency* produces an event that is consumed only by the *Assistance* service of that ambulance. Finally, *Assistance* produces an event for *ECG Analysis* to indicate the status of the current emergency.

Broker-based event-driven interactions use an event bus to manage event queues for a particular IoT application. An event bus is generally implemented using a messaging protocol such as the Advanced Message Queuing Protocol (AMQP) or the Message Queue Telemetry Transport (MQTT). RabbitMQ is the most popular implementation of the AMQP protocol.[2] The EMoNet services shown in Fig. 5(b) interact in the same way as the ones shown in Fig. 5(a), with the fundamental difference that events are now stored in the queue of an event bus.

[1]http://zeromq.org/
[2]https://www.rabbitmq.com/

Event-driven interactions are preferred to direct interactions for implementing Microservice choreographies [9], [23], [11]. Microservices use the strategy *smart endpoints and dumb pipes* [9] to define event-driven interactions in endpoints.

There is an increasing trend to use event-driven interactions for the exchange of data between IoT applications [25], [26]. In fact, the author in [28] found that a vast majority of current IoT platforms provide support for the event-driven interaction schema. In particular, broker-based event driven interactions are gaining considerable attention since MQTT was particularly designed for resource-constrained nodes [29], [30].

D. Exogenous Service Interactions

The exogenous service interaction schema enables a coordinator to define interactions (in the form of a workflow) over mutually anonymous services or other coordinators. Thus, control is always originated in coordinators and services do not interact with each other [31], [32].

Exogenous interactions can be done in one or multiple levels. One-level exogenous interactions are realized by orchestration [33], [34], where the coordinator is a workflow engine running in a specialized server. Fig. 6(a) shows one-level interactions for the EMoNet workflow.

EMoNet Workflow Engine is the coordinator for all the involved services. It passes control to *Heart Rate History* and *ECG Analysis* sequentially, in order to pull and analyze the last sensor readings. Then, according to the results of the analysis, the coordinator decides if there are signs of a heart attack. If so, the coordinator passes control to *Emergency* and *Assistance*, in that order. A glance at Fig. 6(a), reveals that control is always originated in the coordinator, and services are only concerned with returning control and data after performing some computation.

Multi-level exogenous interactions are done by hierarchical orchestration [35], [36] or exogenous connectors [37], [38], [39]. In this schema, multiple coordinators create a hierarchy of service interactions. Unlike, one-level exogenous interactions, in this schema control flows over multiple distributed coordinators.

Hierarchical orchestration [36] has multiple workflow engines, each of them responsible for the interaction of services or other workflow engines. In other words, it allows nesting a workflow within another workflow. Fig. 6(b) depicts a two-level hierarchical orchestration for the EMoNet services. *EMoNet Workflow Engine* coordinates the execution of coordinators *Monitoring Workflow Engine* and *Decision-Making Workflow Engine*. First, *EMoNet Workflow Engine* passes control to *Monitoring Workflow Engine* which is responsible for the interactions of the services *Heart Rate History* and *ECG Analysis*. Once control is returned from *Monitoring Workflow Engine* to *EMoNet Workflow Engine*, the latter passes control to *Decision-Making Workflow Engine*. If *Decision-Making Workflow Engine* determines that there are signs of a heart attack, it passes control to *Emergency* and *Assistance* sequentially. Finally, the control flow ends when the *Decision Making Workflow Engine* returns control to *EMoNet Workflow Engine*. Fig. 6(b) shows that a

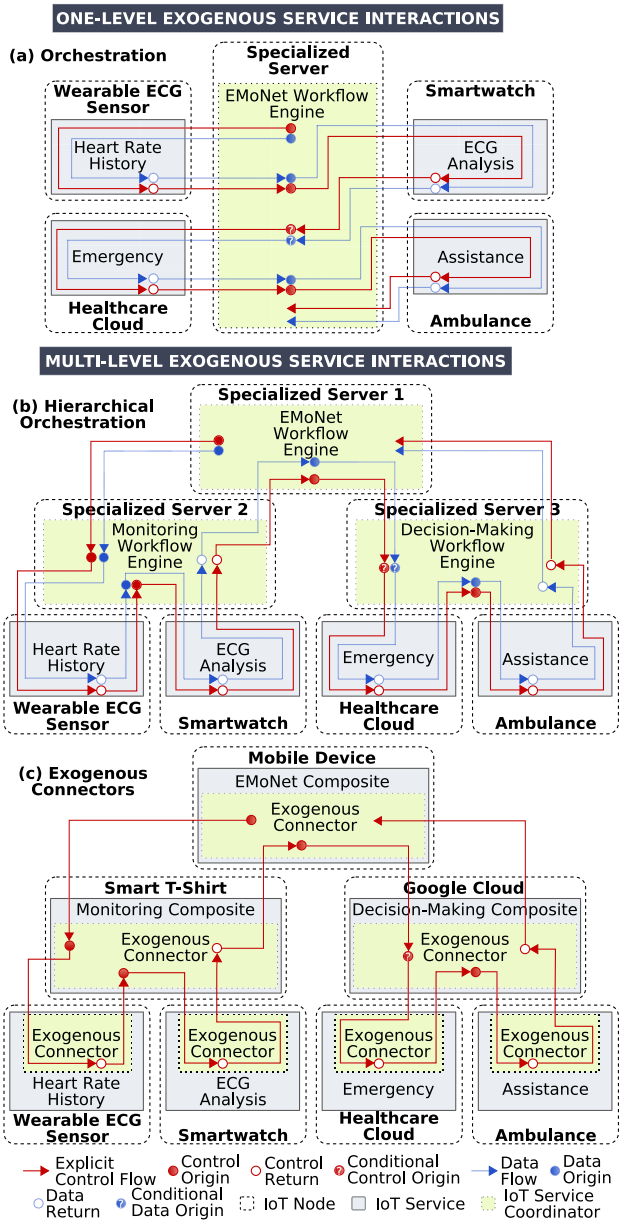

Fig. 6. Exogenous service interactions for the EMoNet workflow.

coordinator is able to pass and receive control and data to and from other coordinators.

Exogenous connectors are lightweight distributed coordinators that define micro-workflows. Fig. 6(c) illustrates how exogenous connectors coordinate service interactions for the EMoNet workflow. The control flow is the same as the one depicted in Fig. 6(b) for hierarchical orchestration. Unlike hierarchical orchestration, where control can be passed from a coordinator to a service, in exogenous connectors control is only passed between cooordinators (as data is an orthogonal dimension). Furthermore, the composition of two services results in a composite service (not a workflow) that preserves all the operations from the sub-services. Another difference is that coordinators do not need any specialized server as they can run in any IoT node (including resource-constrained nodes). For the EMoNet workflow, *Heart Rate History* and *ECG Analysis* are composed into *Monitoring Composite* which is deployed in a smart t-shirt; similarly, *Emergency* and *Assistance* are composed into *Decision-Making Composite* which is deployed in the *Google Cloud*. Exogenous connectors allow composite services to be further composed into even bigger services. For example, the *Monitoring Composite* and the *Decision-Making Composite* are composed into the *EMoNet composite* which is deployed in the patient's mobile device.

Due to the popularity of one-level exogenous interactions in SOA, in the last years we have seen the emergence of software platforms to support such a schema, e.g., Intel IoT SOL (Service Orchestration Layer) [40]. To the best of our knowledge, there are currently no IoT platforms for multi-level exogenous interactions.

IV. Evaluation and Results

This section presents the results of a qualitative evaluation of our service interaction schemas. A tick mark indicates that a specific interaction schema fulfills the requirement being analyzed, while a cross mark means the opposite. *NA* means that the analysis is not applicable for a particular interaction schema. In order to determine which schema best fulfills the scalability requirements of IoT applications, we specifically investigate the following research questions:

- **RQ1:** Which schemas allow the visualization of control flow?
- **RQ2:** Which schemas allow a separate reasoning between control and computation?
- **RQ3:** Which schemas allow decentralized interactions?
- **RQ4:** Which schemas enable services that are unaware of the location of other services?

A. RQ1: Explicit Control Flow vs Implicit Control Flow

Control flow can be *explicit* or *implicit*. Explicit control flow is visible as an entity defines the order in which individual services are executed. Conversely, implicit control flow is opaque since it is not defined anywhere, but it is implicit in the interactions of the participant services. Table I shows that event-driven interactions do not support visible control flow as it is implicit in the collaborative exchange of events (see Fig. 5) [17], [27]. In both direct interactions and indirect interactions, services are the entities who control the application flow, e.g., *ECG Analysis* defines a guard to execute *Emergency* when a heart attack is detected (see Figs. 3 and 4). In exogenous interactions, coordinators define control, e.g., *EMoNet Workflow Engine* defines control structures to realize one-level exogenous interactions for the EMoNet workflow (see Fig. 6(a)).

The amount of service interactions in IoT applications may become overwhelming due to the huge number of nodes involved. Since it is not visible, implicit control flow limits the scalability of IoT applications as the number of services grows and the complexity of service interactions increases.

TABLE I
EXPLICIT CONTROL FLOW IN SERVICE INTERACTION SCHEMAS.

		Explicit control flow
Direct interactions		✓
Indirect interactions		✓
Event-driven interactions	P2P	✗
	Broker-based	✗
Exogenous interactions	One-level	✓
	Multi-level	✓

Implicit control flow has been an issue for software companies over many years and it is undoubtedly a barrier for IoT. For instance, Netflix has recently expressed that implicit control flow limits the scalability of distributed applications, as they found that process flows are spread across multiple applications and it is difficult to monitor workflow processes. As an attempt to visualize control flow, Netflix recently moved from event-driven interactions to exogenous interactions [3].

Visualizing control flow (e.g., to find execution paths) in event-driven interactions is challenging because it is necessary to look at logs to understand the correlation between events [41]. This evidently makes it hard to monitor workflow execution, debug code and modify application workflow. For instance, in the event-based EMoNet workflow it is hard to know which is the most popular ambulance, since there are many ambulances involved. Explicit control flow helps to mitigate this problem so a graphical user interface [3], [42] can be used to display a visual representation of the blueprint with the paths the control has taken during the execution of EMoNet.

In general, explicit control flow is crucial to facilitate the monitoring, maintenance and evolution of IoT applications [23], [3], [27].

B. RQ2: Separation between control and computation

IoT is characterized by heterogeneity in several forms, e.g., different vendors, different hardware and a wide variety of programming languages. For this reason, control and computation should be orthogonal dimensions in every IoT application, in order to enable a flexible integration of services in a heterogeneous environment [43], [44], [45], [4].

Computation is the low-level functionality of an IoT node (provided by a service), and control defines the logic to realize service interactions. The separate reasoning of these concerns enables application developers to focus on the IoT application logic, whilst IoT service developers can focus on the development of efficient service functionality. This separation not only results in reduced time to market, but also reduced software production and maintenance costs.

In both direct interactions and indirect interactions, a sender and a receiver are tightly coupled in terms of control, since control is always originated in the sender's computation. For example, in the EMoNet workflow done by either indirect interactions or direct interactions, *ECG Analysis* is responsible for the conditional control structure that passes control to *Emergency* when a heart attack is detected (see Figs. 3 and 4).

Services mixing control with computation are not reusable, as control flow may vary from one application to another. Suppose we want EMoNet to execute a planning phase after the analysis phase, in order to predict heart diseases and determine heart attacks in real-time. To do so, the *HPC Computing* node, providing the *ECG Planning* service, is introduced. In the EMoNet workflow done by either direct interactions or indirect interactions, both *ECG Analysis* and *ECG Planning* must be changed to accommodate the new requirement. In particular, the conditional control structure is removed from *ECG Analysis* and added into *ECG Planning* which is now responsible for passing control to *Emergency* (when a heart attack is detected). For that reason, *ECG Analysis* is not reused in the new application.

Our analysis of the separation between control and computation is not applicable for event-driven interactions, since control flow is implicit in this schema. Nevertheless, in event-driven interactions, events are originated in service computation (see Fig. 5). For example, *Emergency* and *ECG Planning* would require changes in their computation so as to accommodate the planning phase. In particular, *ECG Planning* needs to consume the events produced by *ECG Analysis*, while *Emergency* needs to consume the events produced by *ECG Planning*. For that reason, *Emergency* is not reused in the new application.

Table II shows that only exogenous interactions separate control from computation, as control is always originated in the coordinator(s) (see Fig. 6). In contrast to the rest of the schemas, exogenous interactions do not require changing any service to support the planning phase, but only changing the application logic defined in the coordinator(s). Thus, as business requirements change, developers can manage changes in the application logic without taking care of IoT service functionality [43].

TABLE II
SEPARATION BETWEEN CONTROL AND COMPUTATION IN SERVICE INTERACTION SCHEMAS.

		Separation between control and computation
Direct interactions		✗
Indirect interactions		✗
Event-driven interactions	P2P	N/A
	Broker-based	N/A
Exogenous interactions	One-level	✓
	Multi-level	✓

When events or control are originated in service computation, an application workflow is embedded in the code of plenty of services. This is in fact one of the reasons for which Netflix stop using event-driven interactions. Exogenous interactions is the only schema that enables the development of workflow-agnostic services, as a consequence of the separation between control and computation. For that reason, Netflix preferred the use of exogenous interactions to event-driven interactions.

C. RQ3: Decentralized Service Interactions

Service interactions can be *centralized* or *decentralized*. Centralized service interactions means that control, events (or even data) pass through a single central entity. By contrast,

decentralized service interactions means that control, events (or even data) are passed in a P2P fashion as workflow (expressed by control or events) is distributed over two or more entities.

Table III shows that indirect interactions, one-level exogenous interactions (i.e., orchestration) and broker-based event-driven interactions are centralized schemas. Indirect interactions require a service bus for passing control and data between services (see Fig. 4). Broker-based event-driven interactions use an event bus to handle events (see Fig. 5(b)). In one-level exogenous interactions, a central engine defines a workflow for passing control (and frequently data) between services (see Fig. 6(a)).

TABLE III
DECENTRALIZATION IN SERVICE INTERACTION SCHEMAS.

		Decentralization
Direct interactions		✓
Indirect interactions		✗
Event-driven interactions	P2P	✓
	Broker-based	✗
Exogenous interactions	One-level	✗
	Multi-level	✓

Even though a centralized approach facilitates the design and maintenance of an IoT application, it possesses several drawbacks that have been recognized by many researchers [35], [4], [46], [36]. For example, in Fig. 6(a) the data generated by *Wearable ECG Sensor* (which is important for *ECG Analysis*) will be routed through *EMoNet Workflow Engine*, even if this data is unimportant to that coordinator. In general, a centralized approach requires an extra network hop for service interactions.

Furthermore, IoT nodes usually generate a huge amount of data. Hence, a central entity may potentially become a performance bottleneck since all the communication will pass through it; thereby, leading to high consumption of network bandwidth, and therefore, unnecessary network traffic. A central entity can also become a single point of failure and attack, thereby impacting the availability of an IoT application.

No single organization should govern an entire workflow or data, as an IoT application may cross administrative domains and organizations may want control over their own part. For example, EMoNet could cross two administrative domains: a data analytics company that processes sensor data and a health telemetry company that monitors patients' heart rate.

According to [5], IoT nodes must possess the ability to interact among themselves with no mediator between them. Decentralized service interactions are more complex than their counterpart, but they bring up increased scalability, availability and reliability for an IoT application by:

- Improving concurrency, load balancing and fault-tolerance due to the use of multiple loci of control or multiple event handlers.
- Bringing performance enhacements (e.g., better throughput) for service interactions.
- Reducing network traffic and latency. as no extra hop is required for service interactions.

Table III shows that decentralization is present in direct interactions, multi-level exogenous interactions (i.e., hierarchical orchestration and exogenous connectors) and P2P event-driven interactions. Direct interactions do not require any mediator for passing control between services (see Fig. 3). In multi-level exogenous interactions, coordinators are the only entities that pass control to services or other coordinators (see Figs. 6(b) and 6(c)). Similarly, P2P event-driven interactions do not rely on a bus for event management, as every service is responsible of its own queue (see Fig. 5(b)).

D. RQ4: Location Transparency

IoT is highly dynamic due to the intermittent connection and spontaneous failures of IoT nodes, resulting in nodes (and ergo services) frequently changing locations over time. For that reason, churn is one of the main challenges of IoT applications as they usually operate in a dynamic and uncertain environment [6], [47]. For example, the *Wearable ECG Sensor* is a resource constrained-node that can run out of battery with the subsequent disconnection from the network. Similarly, an *Ambulance* may experience frequent disconnections due to its high mobility.

Service location transparency is crucial to mitigate churn in IoT applications, as it enables services to be unaware of the physical location of other services. Table IV shows that indirect interactions, broker-based event-driven interactions and exogenous interactions provide location transparency. In indirect interactions, the service bus is the only entity aware of services' locations. In broker-based event-driven interactions, publishers and subscribers do not know the location of one another, but they only know what events to produce and consume, respectively. In exogenous interactions, coordinators encapsulate services' locations as they are responsible for service interactions.

TABLE IV
LOCATION TRANSPARENCY IN SERVICE INTERACTION SCHEMAS.

		Location transparency
Direct interactions		✗
Indirect interactions		✓
Event-driven interactions	P2P	✗
	Broker-based	✓
Exogenous interactions	One-level	✓
	Multi-level	✓

Direct interactions and P2P event-driven interactions do not support location transparency, as they require senders to know the location of receivers a priori. The main problem of these schemas is that senders need to be changed every time the receivers' location change. Although this issue can be solved using a service discovery mechanism (e.g., querying a service registry) [8], it would require an extra network hop. In fact, centralized interaction schemas enclose a discovery component in the middleman [48]. Assuming there is no discovery mechanism for the EMoNet workflow based on direct interactions or P2P event-driven interactions, *Emergency* must be updated every time an *Ambulance* changes location. This is a frightening situation for EMoNet because there is a huge number of ambulances constantly changing locations.

TABLE V
ANALYSIS OF SERVICE INTERACTION SCHEMAS.

	Direct interactions	Indirect interactions	Event-driven interactions		Exogenous interactions	
			P2P	Broker-based	One-level	Multi-level
Explicit control flow	✓	✓	✗	✗	✓	✓
Separation between control and computation	✗	✗	NA	NA	✓	✓
Decentralized control flow	✓	✗	✓	✗	✗	✓
Service location transparency	✗	✓	✗	✓	✓	✓

V. DISCUSSION

Table V summarizes the results of our qualitative evaluation. It particularly shows how well service interaction schemas fulfill the scalability requirements of IoT applications: explicit control flow, separation between control and computation, decentralized interactions, and service location transparency.

Direct interactions and indirect interactions cover 50% of the requirements, respectively. Event-driven interactions is the worst schema since it only meets 25% of those requirements in both P2P and broker-based. Lacking only decentralization, one-level exogenous interactions cover 75% of the desiderata. Multi-level exogenous interactions is the only schema that fulfills all the scalability requirements of IoT applications.

In some scenarios, it could be useful to combine interaction schemas. For example, in order to provide asynchronous interactions in EMoNet, services can combine event-driven interactions with direct interactions. *ECG Analysis* can interact via an event bus with both *Heart Rate History* and *Emergency*, whilst *Emergency* can use direct interactions to request the *Assistance* service of the nearest ambulance.

A service bus can be [21]: (i) *distributed*, (ii) *with technical intelligence* or (iii) *with business intelligence*. Options (i) and (ii) are used only for data and control routing, whilst (iii) can be used to define coordination logic in addition [22]. Even though it is typically used only for straightforward workflows, (iii) is a special case of one-level exogenous interactions.

Although the Microservices community recommends the avoidance of (iii) as they do not want business logic embedded in a service bus [22], there is an increasing tendency to use exogenous interactions for Microservices in traditional SOA applications [34], [3]. By contrast, in the context of IoT, event-driven interactions are currently more popular. However, given the advantages of exogenous interactions, as evidenced by their increasing adoption in traditional SOA applications, we envision that exogenous interactions will increase in popularity in Microservice-based IoT applications in the next years.

A Distributed Service Bus (DSB) [49] is often seen as a decentralized approach due to the existence of a federation of brokers. However, it consists solely of a distribution of middleware components over different nodes. According to our view of decentralization presented in Sec. IV-C, the existence of an intermediary (or intermediaries) for service interactions leads to a centralized approach. As we noted in Sec. IV-C, a purely decentralized approach removes the need of a middleman (or middlemen) which, among other issues, introduces an extra network hop for service interactions.

In order to achieve decentralization, the Microservices community fosters direct interactions between Microservices. Nevertheless, direct interactions impact performance because a connection must be open for the entire duration of an interaction, and a Microservice participant needs a reference (i.e., a client library) for every Microservice it communicates directly with. Maintaining references to other Microservices is costly. Furthermore, a HTTP connection may become a bottleneck, especially for long running Microservices. This is undoubtedly a problem for resource-constrained IoT nodes which do not have communication and storage capabilities to support long-running transactions or to store multiple references. To solve this issue, the Internet Engineering Task Force (IETF) has developed the Constrained Application Protocol (CoAP) [50]. CoAP has been proved to be a simpler and more cost-efficient alternative to HTTP/REST in several IoT scenarios involving resource-constrained nodes [51]. Nevertheless, CoAP does not support the separation between control and computation.

The separation between control and data is also crucial for the scalability of IoT applications. It means that data is never passed alongside control, thereby allowing a separate reasoning between data flow and control flow, which could result in the development of an efficient data exchange approach. For instance, a P2P data exchange can be used to reduce the number of network hops, thereby avoiding network congestion as shown by [46]. The separation between control and data also enables the reuse of data flow without the need of modifying control flow. Hence, data flow and control flow can evolve separately. Exogenous connectors in multi-level exogenous interactions provide semantics for the separation between control and data. Although data typically follows control in orchestration approaches, the separation between control and data has already been done in such approaches [46], [52]. For the EMoNet workflow based on exogenous interactions, we assumed that there is no separation between control and data.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we classified and analyzed service interactions into four schemas, namely direct interactions, indirect interactions, event-driven interactions and exogenous interactions.

We conducted a qualitative evaluation to determine which interaction schema best fulfills the scalability requirements of IoT applications: explicit control flow, decentralized interactions, separation between control and computation, and service location transparency. We showed that multi-level exogenous

interactions is the most promising schema since it meets all the desiderata for the scalability of IoT applications.

Network performance is another aspect that needs to be considered when tackling scalability. We would like to conduct experiments to quantitatively evaluate the throughput of the service interaction schemas presented in this paper.

To the best of our knowledge, there are no IoT platforms based on multi-level exogenous interactions. As this is the most promising schema for IoT, we hope to see its realization in the coming years. In fact, we are currently working on the development of such a platform.

REFERENCES

[1] "Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions)," https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/, 2018.

[2] M. Abbott and M. Fisher, *The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise*, 2nd ed. Addison-Wesley, 2015.

[3] Netflix, "Conductor," https://netflix.github.io/conductor/, 2016.

[4] D. Wutke *et al.*, "Model and infrastructure for decentralized workflow enactment," in *ACM Symposium on Applied Computing*, 2008, pp. 90–94.

[5] S. Roy and C. Chowdhury, "Integration of Internet of Everything (IoE) with Cloud," *Beyond the Internet of Things*, vol. 24, no. 6, pp. 199–222, 2017.

[6] R. Buyya and A. Dastjerdi, *Internet of Things: Principles and Paradigms*. Amsterdam Boston Heidelberg: Morgan Kaufmann, 2016.

[7] J. Soriano *et al.*, "Internet of Services," *Evolution of Telecommunication Services*, vol. 7768, pp. 283–325, 2013.

[8] D. Guinard *et al.*, "Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services," *IEEE Transactions on Services Computing*, vol. 3, no. 3, pp. 223–235, 2010.

[9] M. Fowler and J. Lewis, "Microservices: A definition of this new architectural term," https://martinfowler.com/articles/microservices.html, 2014.

[10] K. Khanda *et al.*, "Microservice-Based IoT for Smart Buildings," in *31st Int. Conference on Advanced Information Networking and Applications Workshop*, 2017, pp. 302–308.

[11] O. Zimmermann, "Microservices tenets," *Comput Sci Res Dev*, vol. 32, no. 3, pp. 301–310, 2017.

[12] R. Dijkman and M. Dumas, "Service-oriented design: A multi-viewpoint approach," *Int. J. Coop. Info. Syst.*, vol. 13, no. 04, pp. 337–368, 2004.

[13] C. Pautasso *et al.*, "Restful Web Services vs. "Big"' Web Services: Making the Right Architectural Decision," in *Proceedings of the 17th International Conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 805–814.

[14] "gRPC," https://grpc.io/, 2018.

[15] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," PhD Thesis, University of California, Irvine, 2000.

[16] A. Barros *et al.*, "Service Interaction Patterns," in *Int. Conference on Business Process Management*, 2005, pp. 302–318.

[17] Q. Sheng *et al.*, "Web services composition: A decade's overview," *Information Sciences*, vol. 280, pp. 218–238, 2014.

[18] T. Ahmed and A. Srivastava, "Service Choreography: Present and Future," in *IEEE Int. Conference on Services Computing*, 2014, pp. 863–864.

[19] I. Nakagawa *et al.*, "Dripcast - Architecture and Implementation of Server-less Java Programming Framework for Billions of IoT Devices," *Journal of Information Processing*, vol. 23, no. 4, pp. 458–464, 2015.

[20] M. Schmidt *et al.*, "The Enterprise Service Bus: Making service-oriented architecture real," *IBM Systems Journal*, vol. 44, no. 4, pp. 781–797, 2005.

[21] N. Josuttis, *Soa in Practice: The Art of Distributed System Design*. O'Reilly Media, Inc., 2007.

[22] C. Pautasso *et al.*, "Microservices in Practice, Part 2: Service Integration and Sustainability," *IEEE Software*, vol. 34, no. 2, pp. 97–104, 2017.

[23] S. Newman, *Building Microservices*, 1st ed. O'Reilly Media, 2015.

[24] L. Gong, "A software architecture for open service gateways," *IEEE Internet Computing*, vol. 5, no. 1, pp. 64–70, 2001.

[25] D. Happ and A. Wolisz, "Limitations of the Pub/Sub pattern for cloud based IoT and their implications," in *Cloudification of the Internet of Things*, 2016, pp. 1–6.

[26] Y. Zhang *et al.*, "Integrating Events into SOA for IoT Services," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 180–186, 2017.

[27] M. Fowler, "What do you mean by "Event-Driven"?" https://martinfowler.com/articles/201701-event-driven.html, 2017.

[28] P. P. Ray, "A survey of IoT cloud platforms," *Future Computing and Informatics Journal*, vol. 1, no. 1, pp. 35–46, 2016.

[29] J. Soldatos *et al.*, "OpenIoT: Open Source Internet-of-Things in the Cloud," in *Interoperability and Open-Source Solutions for the Internet of Things*, 2015, pp. 13–25.

[30] A. Antonić *et al.*, "A mobile crowd sensing ecosystem enabled by CUPUS: Cloud-based publish/subscribe middleware for the Internet of Things," *Future Generation Computer Systems*, vol. 56, pp. 607–622, 2016.

[31] K.-K. Lau *et al.*, "Exogenous Connectors for Software Components," in *8th Int. Conference on Component-Based Software Engineering*, 2005, pp. 90–106.

[32] S.-S. Jongmans *et al.*, "Orchestrating web services using Reo: From circuits and behaviors to automatically generated code," *Service Oriented Computing and Applications*, vol. 8, no. 4, pp. 277–297, 2014.

[33] C. Lee *et al.*, "Blueprint Flow: A Declarative Service Composition Framework for Cloud Applications," *IEEE Access*, vol. 5, pp. 17 634–17 643, 2017.

[34] S. Alpers *et al.*, "Microservice Based Tool Support for Business Process Modelling," in *IEEE 19th Int. Enterprise Distributed Object Computing Workshop*, 2015, pp. 71–78.

[35] G. Chafle *et al.*, "Decentralized Orchestration of Composite Web Services," in *13th International World Wide Web Conference (WWW '04)*, 2004, pp. 134–143.

[36] W. Jaradat *et al.*, "Towards an autonomous decentralized orchestration system," *Concurrency Computat.: Pract. Exper.*, vol. 28, no. 11, pp. 3164–3179, 2016.

[37] D. Arellanes and K.-K. Lau, "Exogenous Connectors for Hierarchical Service Composition," in *10th IEEE International Conference on Service Oriented Computing and Applications (SOCA '17)*, 2017, pp. 125–132.

[38] D. Arellanes and K. K. Lau, "D-XMAN: A Platform For Total Compositionality in Service-Oriented Architectures," in *7th IEEE International Symposium on Cloud and Service Computing (SC2 '17)*, 2017, pp. 283–286.

[39] D. Arellanes and K.-K. Lau, "Algebraic Service Composition for User-Centric IoT Applications," in *3rd International Conference on Internet of Things*, 2018.

[40] Intel, "IoT Services Orchestration Layer," http://01org.github.io/intel-iot-services-orchestration-layer, 2016.

[41] A. Burattin *et al.*, "Control-flow discovery from event streams," in *IEEE Congress on Evolutionary Computation*, 2014, pp. 2420–2427.

[42] "Zipkin," https://zipkin.io/, 2018.

[43] "Amazon Simple Workflow Service (SWF)," https://aws.amazon.com/swf/, 2018.

[44] F. Leymann, "Web Services: Distributed Applications Without Limits." in *Database Systems for Business, Technology and Web*, vol. 26, 2003, pp. 2–23.

[45] K.-K. Lau *et al.*, "A Component Model for Separation of Control Flow from Computation in Component-Based Systems," *Electronic Notes in Theoretical Computer Science*, vol. 163, no. 1, pp. 57–69, 2006.

[46] A. Barker *et al.*, "The Circulate architecture: Avoiding workflow bottlenecks caused by centralised orchestration," *Cluster Computing*, vol. 12, no. 2, pp. 221–235, 2009.

[47] P. Barnaghi and A. Sheth, "On Searching the Internet of Things: Requirements and Challenges," *IEEE Intelligent Systems*, vol. 31, no. 6, pp. 71–75, 2016.

[48] A. Ngu *et al.*, "IoT Middleware: A Survey on Issues and Enabling Technologies," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 1–20, 2017.

[49] F. Baude *et al.*, "ESB federation for large-scale SOA," in *ACM Symposium on Applied Computing*, 2010, pp. 2459–2466.

[50] Z. Shelby *et al.*, "Constrained Application Protocol (CoAP)," https://datatracker.ietf.org/doc/rfc7252/, 2014.

[51] "Comparing the cost-efficiency of CoAP and HTTP in Web of Things applications," *Decision Support Systems*, vol. 63, pp. 23–38, 2014.

[52] C. Pautasso, "Composing RESTful services with JOpera," in *8th Int. Conference on Software Composition*, 2009, pp. 142–159.