Future Generation Computer Systems 108 (2020) 827–848

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Evaluating IoT service composition mechanisms for the scalability of IoT systems

Damian Arellanes *, Kung-Kiu Lau

Department of Computer Science, The University of Manchester, Manchester M13 9PL, United Kingdom

ARTICLE INFO

Article history:
Received 29 July 2019
Received in revised form 28 January 2020
Accepted 24 February 2020
Available online 12 March 2020

Keywords:
Scalability
IoT service composition
Orchestration
Choreography
Dataflows
Workflows

ABSTRACT

The Internet of Things (IoT) is an emerging paradigm where practically every (physical and virtual) thing will be interconnected through innovative distributed services. Since the number of connected things is rapidly growing, IoT systems will require the composition of plenty of services into complex workflows. Thus, scalability in terms of the size of IoT systems becomes a significant concern. In this paper, we review and evaluate the fundamental semantics of existing IoT service composition mechanisms to determine how well they fulfil the scalability requirements of IoT systems. We identify scalability desiderata and, accordingly, our findings show that dataflows, orchestration and choreography do not fully satisfy such desiderata, unlike a novel composition mechanism called DX-MAN.

1. Introduction

The Internet of Things (IoT) is an emerging paradigm that promises the interconnection of (physical and virtual) things through innovative distributed services. Like traditional enterprise services, IoT services interact in many different ways via the Internet, in order to realise a global system workflow. However, unlike traditional enterprise systems, IoT systems will require the interaction of billions of services as the number of connected things (and therefore services) is rapidly growing [1,2]. Thus, scalability becomes a crucial concern.

Scalability is typically considered as a system capability to handle increasing workloads [3–8]. In particular, vertical scalability [9–11] refers to the addition or removal of computing resources in a single IoT node, while horizontal scalability [2,12,13] involves the addition or removal of IoT nodes. These kinds of scalability have been addressed by an extensive body of research [4,10,11,14–20], unlike scalability in terms of the number of services composed in an IoT system, which we refer to as *functional scalability*.

Existing service composition mechanisms were primarily designed for the integration of static enterprise services, not for the physical world. For that reason, they may not address the functional scalability challenges that IoT systems pose. Early IoT systems were deployed in closed environments, using private Application Programming Interfaces (APIs) and private data. However, future IoT systems will be deployed in open environments (also known as software ecosystems [21]) with an overwhelming number of available services, as a result of the huge number of connected things [22]. For that reason, billions of IoT services will be composed into complex IoT systems [23–28]. This raises the challenging question of *How to construct IoT systems composed of an ultra-large number of services?*

In this paper, we study the ability of service composition mechanisms to handle any number of IoT services. Our aim is to determine *which service composition mechanism best fulfils the functional scalability requirements of IoT systems*. To answer this, we have reviewed the fundamental semantics (not specific implementation technologies) of existing composition mechanisms, and proposed an evaluation framework that considers six crucial desiderata: (i) explicit control flow [29,30]; (ii) distributed workflows [23,31,32]; (iii) location transparency [20,33]; (iv) decentralised data flows [23,32,34]; (v) separation of control, data and computation [35–38]; and (vi) workflow variability [23,39–41]. Our evaluation framework serves as a guideline for defining the semantics of future IoT service composition mechanisms.

The remainder of the paper is structured as follows. In the next section, we present an overview of IoT services, scalability and service composition. Section 3 outlines the related work on IoT service composition mechanisms. Section 4 introduces a motivating IoT scenario based on a smart parking system. The scenario is then considered in Section 5 to comprehensively describe the requirements for functional scalability and the rationale behind them. Section 6 analyses service composition mechanisms on the basis of the functional scalability requirements. Section 7 presents an evaluation that determines how well service composition mechanisms fulfil the scalability requirements. Our findings

* Corresponding author.
E-mail addresses: damian.arellanesmolina@manchester.ac.uk (D. Arellanes), kung-kiu.lau@manchester.ac.uk (K.-K. Lau).

https://doi.org/10.1016/j.future.2020.02.073
0167-739X/

are then discussed in Section 8. Finally, Section 9 draws the conclusions and presents the directions for future research.

2. Background

This section presents a background for the rest of the paper, which includes an overview of IoT services, service composition and scalability.

2.1. IoT services

Kevin Ashton coined the term *Internet of Things* (IoT) in a presentation made in 1999 at Procter and Gamble [42], referring to the interconnection of everything via the Internet for the creation of an ubiquitous computing environment [43]. As per the recommendation of ITU-T Y.4000, IoT has been recently redefined as "*a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies*" [44].

A *thing* is practically a physical or virtual construct of the real-world, which is capable of being identified and integrated into communication networks through specific protocols. The difference between physical and virtual things lies in their tangibility [45]. A physical thing is a tangible object of the physical world, which is capable of being sensed, actuated and connected, e.g., home appliances, robots, buildings, plants and people. Contrastingly, a virtual thing is a non-tangible construct formed from a human idea which only exists in the information world, e.g., Clouds and enterprises.

The broad range of available things inevitably requires dealing with a high degree of heterogeneity in a distributed environment. Accordingly, a Service-Oriented Architecture (SOA) represents the best way of dealing with this issue [46]. It is *a logical way of designing a software system to provide services either to end-user applications or other services distributed in a network, via published and discoverable interfaces* [47]. Thus, it is expected that physical and virtual things will provide services to expose behaviour via interfaces [5,48–52].

According to the Oxford dictionary, the word *service* can be a noun or a verb referring to the *action of helping or doing work for someone*. Considering a service as a noun allows the encapsulation of behaviour (i.e., actions) in the form of operations described as verbs. Thus, *a software service is a distributed software component that provides a set of operations through network-accessible endpoints* [53,54]. In general, IoT services are virtual representations of the behaviour of things, which can be combined with other services into more complex behaviours to yield complex service-oriented IoT systems [52,55–57]. Thus, things are integrated through the composition of the services they provide (see Section 2.2).

Resource-constrained things (e.g., pulse sensors) typically provide fine-grained services for basic functionality (e.g., fetching sensor data), whilst non-resource constrained things (e.g., a Cloud) may offer coarse-grained services in addition (e.g., services for geolocation or complex industrial processes). Enterprise services are typically coarse-grained as they are deployed on resource-rich infrastructures, whilst services of physical things are often fine-grained because they are usually deployed on resource-constrained things.

Fig. 1 shows the relationship between things, services and operations. Fig. 1(a) depicts a *washing machine* (i.e., a resource-constrained physical thing) that offers the *Washing* and *Drying* fine-grained services with two operations each (for starting and stopping the respective processes). Fig. 1(b) shows a *City Council Cloud* (i.e., a non-resource constrained virtual thing) that offers the services *CouncilTax* and *Parking*. The *CouncilTax* service provides the operations *pay* (to pay a tax bill) and *check* (to query council tax information). The *Parking* service offers the operations *getNearest* (for getting the closest parking space from a driver's location) and *registerDisabled* (for registering an impaired driver).

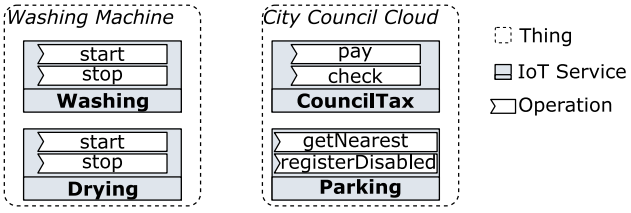

Fig. 1. Relationship between things, IoT services and operations.

For the rest of the paper, we use the notation *S.O* to denote an operation *O* in service *S*, e.g., *Parking.getNearest* refers to the *getNearest* operation provided by the *Parking* service.

2.2. IoT service composition revisited

An IoT service is a distributed unit of composition, which constitutes the virtual representation of a thing's behaviour, and can be either atomic or composite. An *atomic service* is a well-defined and self-contained piece of behaviour that cannot be divided into other services [58–60]. A *composite service*, on the other hand, is a more complex unit that provides value-added functionality and is formed by the combination of (atomic or composite) services [52,55,57–59,61,62]. For example, a humidity sensing service can be combined with a temperature service into an air conditioning composite [63].

The ability of combining services is referred to as *compositionality* and is realised by a *composition mechanism* [59]. Thus, an IoT system requires a things infrastructure, the definition of what a service is and the selection of a composition mechanism [64]. In any scenario, composition is done regardless of both the technologies being used and the things infrastructure. Service technologies include REST [65,66], WS-* [67,68], OSGi [69,70] and many others.[1]

A service composition mechanism defines a meaningful interaction between services [59] by considering two functional dimensions: control flow and data flow [71,72]. Control flow refers to the order in which interactions occur [73,74], whilst data flow defines how data is moved among services [71]. In this paper, we focus on service composition mechanisms that define behaviour by workflows, namely (centralised and distributed) dataflows, (centralised and distributed) orchestration, choreography, and a novel composition mechanism called DX-MAN. Section 6 provides a detailed description of these mechanisms.

A workflow describes a series of discrete steps for the realisation of a computational activity, which can be control-driven, data-driven or hybrid [75]. In a control-driven workflow, steps (also known as tasks [76], actors [77], transitions [78], procedures [79], thorns [80], activities [81] and units [82]) are executed according to explicit control flow constructs that define sequencing, looping, branching or parallelising. A data-driven workflow invokes steps whenever data becomes available without explicitly defining any control flow constructs [83]. In a hybrid workflow, some parts are control-driven, while others are data-driven [82]. Fig. 2 illustrates a generic workflow that executes the operation op_1, then decides to invoke either op_2 or op_3 and, finally, triggers the operations op_4 and op_5 in parallel. Operation invocations happen regardless of the workflow kind.

[1] In RESTful services, operations are exposed as resources [65].

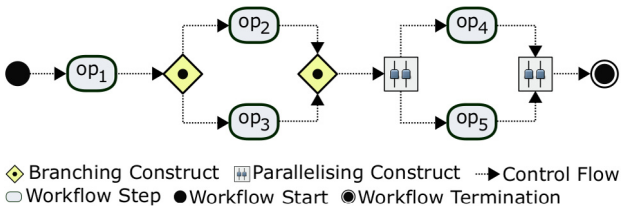

Fig. 2. A generic workflow.

Workflows are increasingly important for IoT systems because they allow the integration of IoT services into complex tasks that automate a specific context [84–88]. For example, a smart home can be automated with a workflow that regulates the temperature of a room according to environmental changes. In the domain of smart agriculture, a workflow can be defined to analyse data coming from harvest sensors, predict diseases and react accordingly.

2.3. Scalability of IoT systems

With the advent of hardware technologies, the number of IoT services is rapidly growing due to the excessive increase in the number of connected things. Currently, there are about 19 billion connected things, and it is predicted that this number will grow exponentially in the coming years [1,2]. Thus, unlike traditional enterprise systems, scalability becomes a crucial concern for the full realisation of IoT systems.

Typically, scalability is the capability to handle increasing workloads in a IoT system [3–8]. In this case, it is a metric that indicates how system performance improves over time. Workloads are typically measured in terms of either the number of requests dispatched [3] or the data streams generated [7]. The overall goal of scalable solutions is to enhance the Quality of Service (QoS) for guaranteeing a certain level of performance under the presence of high workloads, e.g., by minimising bandwidth, energy, latency and response time while maximising throughput. To quantitatively measure QoS, several network aspects of a service are considered such as jitter, throughput, packet loss and availability [8].

Currently, there are two kinds of scalability: vertical and horizontal.[2] Vertical scalability (or scaling up) [9–11] refers to the addition or removal of computing resources in a single thing, e.g., adding more memory to increase buffer size or adding more processor capacity to speed up processing. On the other hand, horizontal scalability (or scaling out) [2,12,13] involves the addition or removal of things in an IoT system. Its goal is to distribute the workload over multiple things to decrease individual loads, minimise response time and enhance concurrency. Fig. 3 depicts the contrast between vertical and horizontal scalability.

Both vertical and horizontal scalability have been extensively addressed in the literature [4,10,11,14–20], unlike *functional scalability* which we refer to as the capability to accommodate growth in terms of the number of services composed in an IoT system (see Fig. 3(c)). In particular, it enables the composition of any number of services, without severely impacting global system properties such as performance, maintenance, evolution and monitoring. Hence, functional scalability is crucial for dealing with IoT systems composed of billions of services. Fig. 4 shows that it is orthogonal to vertical and horizontal scalability.

[2] IoT cloud environments benefit from dynamically scaling vertically, horizontally or both.

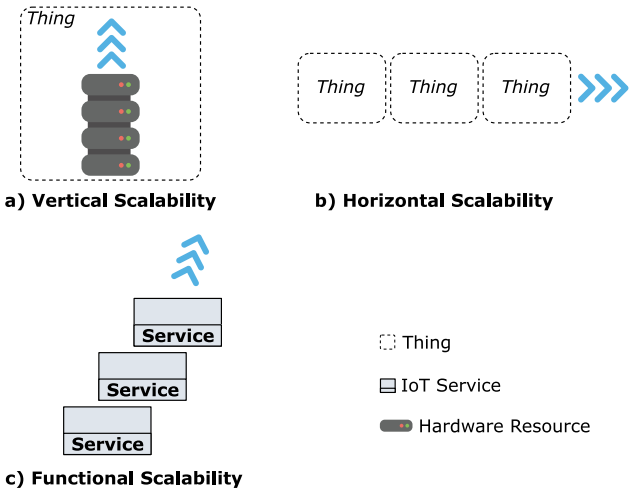

Fig. 3. Scalability of IoT systems.

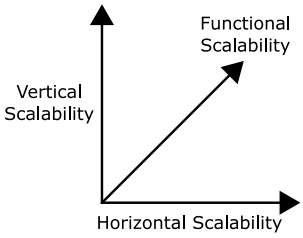

Fig. 4. Scalability dimensions.

Like the other kinds of scalability, functional scalability requires the definition of metrics to measure the degree of satisfaction for accommodating new services. In this paper, we propose six qualitative metrics which we discuss in Section 5. We do not claim that such metrics are complete since quantitative metrics for QoS, identified for vertical and horizontal scalability, can also be important. However, quantitative metrics are only applicable to specific implementations. As service composition is an abstraction rather than a concrete implementation, we strongly argue that qualitative metrics are the best ones to measure the degree of satisfaction of functional scalability in service composition mechanisms. For the rest of the paper, the terms scalability and functional scalability are used interchangeably.

3. Related work

This section presents the current IoT service composition mechanisms that define behaviour by workflows, namely (centralised and distributed) dataflows, (centralised and distributed) orchestration, choreography, and a novel composition mechanism called DX-MAN. We particularly focus on the fundamental semantics of these mechanisms instead of addressing specific implementation technologies. This is because semantics constitutes general theory that defines how to compose services conceptually rather than a concrete implementation (that can only be evaluated in specific scenarios). Significantly, fundamental semantics underlies so-called composition algorithms [8,89–102], programming frameworks [84,103–108], languages [109,110] and platforms [111–117], which have been somehow confusingly included in existing "IoT service composition" surveys [3,5,24,118,119].

It is also essential to mention that IoT service composition is just another name for traditional SOA composition and it is done

regardless of so-called service "architectures" such as the ones defined for Microservices. Microservice Architecture [120–122] has gained considerable attention in the last few years and is becoming increasingly important and popular for the development of IoT systems [123–127]. Every Microservice Architecture is an SOA, but not the other way round [128]. Hence, the service composition mechanisms presented in this sub-section can be used interchangeably in both Microservices and traditional SOA services. In fact, there are no composition mechanisms specifically aimed for Microservices.

Dataflows, or *Flow-Based Programming* [129–131], is a composition mechanism that defines implicit workflows as directed graphs where vertices receive input data streams, carry out some computation and pass the result to other vertices via an edge for further processing [71,132]. A centralised approach [83,116] fully coordinates the exchange of data streams exogenously (i.e., from outside services), whereas a distributed one [37,38,133,134] partitions a complex workflow dataflow over multiple coordinators. Currently, dataflows is the most popular IoT service composition mechanism, so there are plenty of available technologies for defining data-driven IoT workflows [83,126,135].

Orchestration defines explicit workflow control flow structures to coordinate the invocation of service operations exogenously and, like dataflows, it can be centralised or distributed. A centralised orchestration [72,136–138] has full control over all the services composed, whereas a distributed approach (also known as "decentralised orchestration" [139–148]) defines sub-workflows for a collaborative exchange of workflow control flow over the network. Although orchestration is not as popular as dataflows, there are some implementations available for this mechanism to support the definition of control-driven IoT workflows [149–151].

Choreography [136–138,152,153] is another composition mechanism that defines workflow control flows for the invocation of operations in services. Unlike orchestration, it relies on a public protocol which specifies a global "service conversation" via decentralised interactions, and it is modelled using a choreography modelling language [122,138,153]. When a choreography is enacted, the composed services exchange control according to the protocol to realise decentralised workflows. It is because of this decentralised nature that there is an increasing trend of implementing technologies for choreographing IoT services [154–156]. Notably, some of these technologies (e.g., [155]) implement choreographies based on the data-driven paradigm with no notion of public protocols, which do not define any composite service but a bunch of interactions via events or decentralised message passing. So, they do not follow the standard definition as noted by [35].

DX-MAN [157–159] is a model that uses exogenous composition operators [160,161] to algebraically compose IoT services in a hierarchical bottom-up manner. The result of composition is not a workflow, but an IoT composite service which is semantically equivalent to a potentially infinite family of (explicit) workflow control flows. As DX-MAN takes the best properties from choreography and orchestration, it enables decentralised data exchanges over the network while decoupling services via (exogenous) workflow control flows [162]. Currently, there is only one platform available to support the definition of algebraic IoT systems [163].

To continue the discussion, Section 6 describes the above composition mechanisms in detail and presents a concrete analysis in terms of functional scalability requirements.

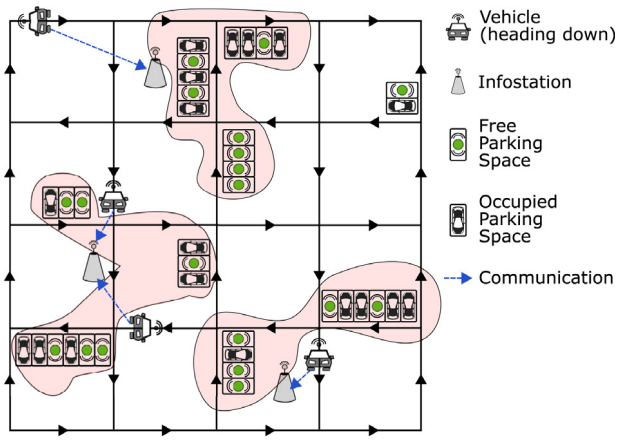

Fig. 5. A smart city with *SPark*.

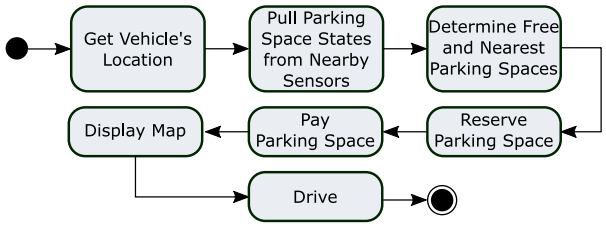

Fig. 6. *SPark* workflow.

4. A large-scale IoT scenario: Smart Parking System (SPark)

To illustrate the context for scalability requirements, this section presents a large-scale IoT scenario in the smart parking domain. The scenario tackles a common problem of large cities and is described as follows.

With the increase of population, large cities have had to deal with daily traffic congestion caused by drivers actively searching for parking spaces, especially during rush hours. As a consequence, there are increased carbon emissions as well as waste of commuters' time and money [164,165]. This section presents a large-scale smart parking system, *SPark*, for self-driving vehicles which efficiently find (and reserve) the nearest parking space in a smart city. *SPark* thus helps to improve parking space utilisation, shorten parking search time, reduce environmental pollution, minimise parking costs and fuel consumption, and alleviate traffic congestion [164].

SPark operates in a smart city with plenty of parking spaces equipped with occupancy sensors whose data is managed by Infostations. Although it is an urban infrastructure device able to collect up-to-date occupancy status from all sensors in range, an infostation only pulls data from the sensors near a vehicle. Fig. 5 illustrates *SPark* with four self-driving cars, where a vehicle gets its location and requests a parking space from the nearest Infostation. The InfoStation then pulls data from the nearby sensors to determine the nearest parking space that is free. To avoid two different vehicles chasing the same parking space, the space is reserved and paid for in advance. Finally, the vehicle displays the desired route and drives towards the selected parking space. Fig. 6 depicts the general workflow of SPark.

To achieve its goal, *SPark* composes a huge amount of IoT services distributed across a city. It is then an ultra large-scale system [23–26] because sub-systems (i.e., services) may potentially integrate other sub-systems (i.e., services) and so on. Fig. 7 depicts the relationship between services and sub-services in

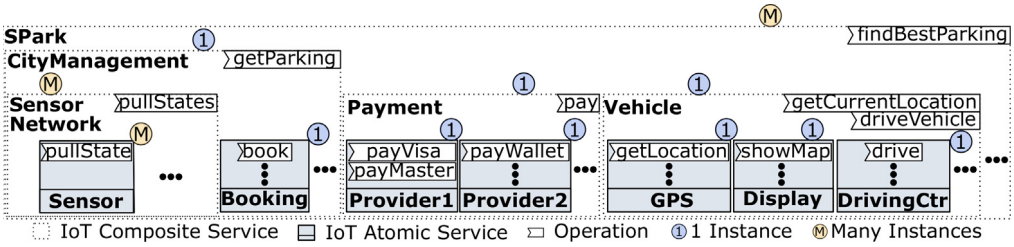

Fig. 7. *SPark* services.

SPark. We distinguish between *atomic services* and *composite services*. An atomic service is the most primitive unit with no internal structure, whilst composite services integrate sub-services and can be integrated into even more complex composites.

To hide complexity and protect behaviour integrity, we assume that the composite services of our example are encapsulated. As IoT composite services potentially reside on different business domains [166], *SPark* cannot be "decomposed" using a top-down approach, but it should be composed in a bottom-up fashion by reusing existing compositions [27,31]. Fig. 7 shows the resulting hierarchical service relationship.

Note that ellipses indicate that services may provide multiple operations or compose many other services. Also note that, in practice, atomic services can be defined as composite services. For example, *DrivingCtr* could be a complex service that integrates services for planning a path and controlling a vehicle. As another example, the *Display* service may internally use a web mapping service and a visualisation service for displaying a route. An alternative (larger) view of *SPark* can be found in Appendix A. For clarity, this section only shows a simplified version of *SPark* which is incrementally described below.

The *SensorNetwork* composite is a wireless network that composes a group of dedicated atomic *Sensor* services. It provides the *SensorNetwork.pullStates* operation as an interface for collecting parking space status in parallel. The collection is done by invoking the *Sensor.pullState* operations from the sensors near the vehicle.

The *CityManagement* composite service composes multiple *SensorNetwork* composites and one atomic *Booking* service. It provides the *CityManagement.getParking* operation for finding and reserving the best (i.e., the free and nearest) parking space. To do so, it collects sensor states with the *SensorNetwork.pullStates* operation, and then determines which parking spaces are free. Finally, it reserves the nearest free parking space using the *Booking.book* operation.

The *Payment* composite is an online electronic payment service that composes two payment providers: *Provider1* and *Provider2*. It chooses which payment method to use when the operation *Payment.pay* is invoked. On the one hand, *Provider1* is an atomic service with the operations *payVisa* and *payMaster* (for paying with Visa or Mastercard). On the other hand, *Provider2* is an atomic service with the operation *payWallet* (for paying with an eWallet).

The *Vehicle* composite encompasses two different behaviours, and composes the atomic services *GPS*, *Display* and *DrivingCtr*. The *Vehicle.getCurrentLocation* operation is an interface for the behaviour of *GPS.getLocation*, which gets geospatial positioning information. The *Vehicle.driveVehicle* operation is more complex, as it invokes *Display.showMap* so as to plan, compute and visualise the route on the vehicle's display. Then, it executes *DrivingCtr.drive* to autonomously drive the vehicle towards the desired parking space.

SPark is the most complex composite service, since it composes the services *CityManagement*, *Payment* and *Vehicle*, and provides the *SPark.findBestParking* operation for the encapsulation of the whole system's behaviour. The behaviour is a workflow for the sequential invocation of *Vehicle.getCurrentLocation*, *CityManagement.getParking*, *Payment.pay* and *Vehicle.driveVehicle*. Appendix B describes the complete workflow of our scenario.

Table 1
SPark service distribution.

Thing	Composite services	Atomic services
CityCouncilCloud	CityManagement, Payment	Booking
ParkingSpace		Sensor
Infostation	SensorNetwork	
ProviderServer1		Provider1
ProviderServer2		Provider2
Vehicle	SPark, Vehicle	GPS, Display, DrivingCtr

As the *SPark* workflow spans multiple administrative domains, our motivating example requires the distribution of services across different geographical locations. Table 1 shows a possible physical deployment configuration. In particular, *CityCouncilCloud* is a virtual thing where services *CityManagement*, *Payment* and *Booking* are deployed. *ParkingSpace* is a physical thing whose occupancy status is provided by the *Sensor* service. *Infostation* is a physical thing that collects data from sensors within a wireless range, via the *SensorNetwork* service. *ProviderServer1* and *ProviderServer2* are physical things of different payment provider companies, where services *Provider1* and *Provider2* are deployed. *Vehicle* is a physical thing that moves around the city, searching for a parking space, which provides the services *SPark*, *Vehicle*, *GPS*, *Display* and *DrivingCtr*. Note that deploying a composite service (e.g., *SPark*) does not necessarily mean that the composed services should be deployed on the same thing. This is because IoT services operate on different administrative domains.

5. Functional scalability requirements

This section presents the functional scalability requirements of IoT systems in terms of *SPark*. These requirements were derived from an extensive review of the literature on large-scale IoT systems and serve as the foundation of our evaluation framework. The requirements are: (i) explicit control flow [29,30]; (ii) distributed workflows [23,31,32]; (iii) location transparency [20,33]; (iv) decentralised data flows [23,32,34]; (v) separation of control, data and computation [35–38]; and (vi) workflow variability [23, 39–41].

5.1. Explicit control flow

Control flow defines the execution order of composed services [71], according to sequencing, branching or parallelising constructs [75]. It is explicit when there is a visible construct

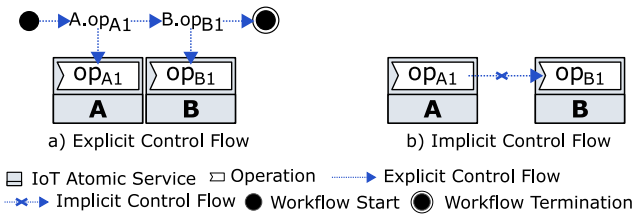

Fig. 8. Explicit control flow vs implicit control flow.

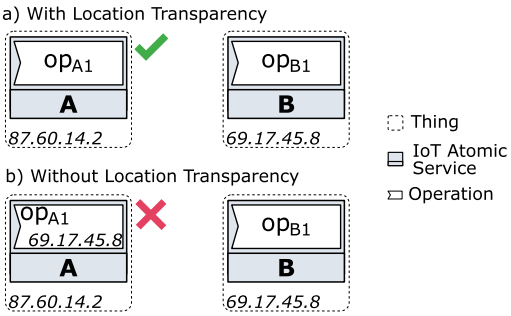

Fig. 9. Location transparency.

defining control flow in a service composition, whilst it is implicit when it needs to be inferred from the collaborative interaction of the composed services [71,74,138].[3] Fig. 8 describes the difference between such concepts. The left side depicts visible constructs for the sequential execution of $A.op_{A1}$ and $B.op_{B1}$. Contrastingly, the right side does not show any control flow constructs, as the workflow logic is hardcoded in the computation of the composed services.

The number of composed services may become overwhelmingly large because of both the huge number of things involved and the complexity of workflow control flow. Consequently, execution failures become unavoidable, more challenging to manage and may potentially unleash catastrophic consequences (for individuals or societies) [167]. For that reason, explicit control flow becomes crucial to tackle functional scalability, since it facilitates monitoring, tracking, verification, maintenance and evolution of large-scale IoT composite services [29,30,122,168,169]. For instance, we can leverage explicit control flow to detect abnormalities in a SPark execution [170–172], or we could easily obfuscate workflow control flow in order to avoid malicious reverse-engineering [173,174].

Imagine that SPark suddenly stops working because of a bottleneck in some service, so developers want to analyse the system execution flow to find out where the problem is. For this, they can leverage explicit control flow to display a visual representation of the execution paths that SPark has taken [29,74,175,176]. By looking at the blueprint, developers can identify the services that perform poorly and react accordingly.

Implicit control flow has historically been a barrier for functional scalability since it hinders control flow visualisation, especially when the number of services increases [29,30,121,122,177,178]. Although there are attempts to visualise control flow [178–182], transforming implicit control flow into an explicit one is still a challenging task, especially when the system workflow is complex. This issue has made software development companies stop composing services with implicit control flow. Netflix [183] is the most prominent case, which has particularly expressed its concern for monitoring composite service executions when control flow is implicit. To tackle this, it has recently developed Conductor [29] to move from compositions with implicit control to compositions with explicit control.

5.2. Location transparency

Large-scale IoT systems are inherently dynamic and uncertain due to the presence of churn in the operating environment [20,25,27,33,184–186]. Churn means that things (and their services) dynamically connect and disconnect from the network as a result of auto-scaling, software upgrades, failures, poor connection and mobility. It is particularly evident when an IoT system uses resource-constrained things with a poor connection [187], or when there are mobile things involved [185,188]. Hence, churn results in physical service locations (i.e., IP addresses) changing over time frequently. For example, the Spark's *Sensor* service is a resource-constrained thing with limited connectivity, which is likely to experience network disconnections. Similarly, the service *Vehicle* may change its IP address because of its high mobility. The same happens to *Provider1*, *Provider2* and *Booking* as they are deployed on the Cloud.

To deal with churn, IoT systems require location transparency, in order to ensure that atomic services are unaware of the physical location of other services [74]. Fig. 9 illustrates this concept. In particular, the lower section shows that the (non-static) IP address of $B.op_{B1}$ is hardcoded in $A.op_{A1}$. The upper section shows a scenario where there is support for location transparency.

Location transparency is typically achieved with a service discovery mechanism that dynamically queries a central service registry [52]. In client-side discovery, service providers register at startup in a registry which is later queried by service consumers. In server-side discovery, a router (e.g., a service bus like a gateway [112]) queries the registry and forwards requests to an available service provider on behalf of service consumers.

A service composition with no location transparency requires service consumers to know the location of service providers in advance. For example, without any support for location transparency, *SPark* would have to be updated every time a *Vehicle* changes location. Nonetheless, this is worrying because there is a massive number of vehicles constantly moving around a city and, therefore, changing location. Addressing this issue with a service discovery mechanism entails the addition of "intrusive" elements to the composition (e.g., a service registry or a naming service) which are not part of the semantics of a composition mechanism. In fact, an atomic service computation would be tightly coupled with the "intrusive" element, since the former needs to be updated whenever the location of the latter changes. We shall keep this in mind in Section 6.

5.3. Distributed workflows

IoT service composition can define a *centralised* or a *distributed* workflow. It is centralised when a single entity governs the entire workflow control flow. Contrastingly, it is distributed when multiple entities collaboratively define control flow [74]. Fig. 10 illustrates the contrast between a centralised and a distributed workflow. The upper part shows a central composite service that defines a workflow for the sequential invocation of $A.op_{A1}$, $A.op_{B1}$, $A.op_{C1}$, $A.op_{D1}$ and $A.op_{E1}$. The lower part shows a possible distribution of the same workflow control flow over three composite services deployed on different things.

Although a centralised workflow facilitates the design and maintenance of an IoT system, implementing a large-scale IoT system using such an approach is inefficient and infeasible [133].

[3] For example, control flow is explicit in imperative languages and implicit in declarative ones.

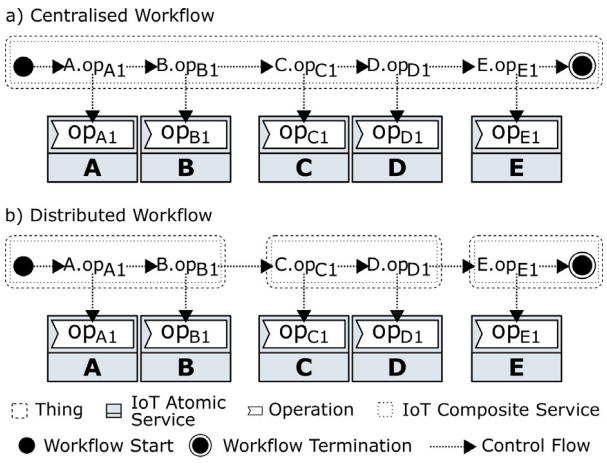

Fig. 10. Centralised workflow vs distributed workflow.

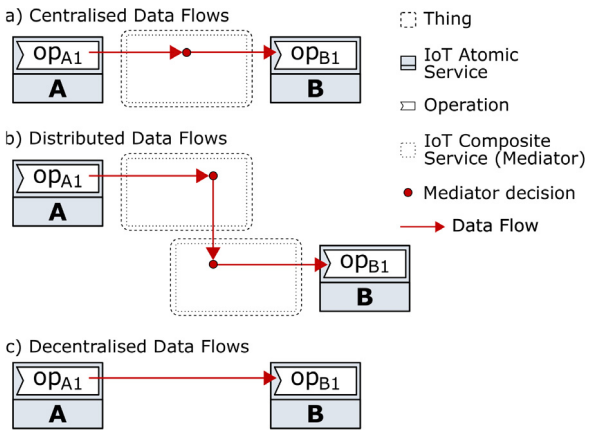

Fig. 11. Data flow approaches.

This is because a central entity constitutes a single point of failure and attack, and leads to a performance bottleneck since all control and data goes through it [27,34,140,142,144,148,189–191]. Moreover, a central entity negatively impacts the availability of an IoT system.

The distributed nature of IoT requires control and data to pass through geographically dispersed entities (potentially deployed on different business domains) [32,37,133,166]. For that reason, no single domain should govern the entire composition workflow, since multiple domains (perhaps in the order of millions) may want control over their own workflow part [74,190,192]. For instance, the *CityManagement* composite may potentially be controlled by the City Council, and a payment provider company could operate the *Payment* composite. This issue clearly implies that every administrative domain should be able to manage their own workflow definition. Thus, a *distributed workflow* should be part of any IoT service composition mechanism, since it allows interoperability between different domains and improves load balancing, throughput and availability of an IoT system [74,87].

5.4. Decentralised data flows

Data flow defines the way data elements are moved from one service to another in a service composition [71]. This process can be done in three different ways: *centralised*, *distributed* or *decentralised*.

In the centralised approach [112,136,193,194], a single composite mediates the exchange of data between services (see Fig. 11(a)). Although it facilitates data management, the mediator becomes a potential bottleneck and introduces additional network hops for data exchange, since all data passes through it. This negatively impacts QoS by increasing the response time, and leads to network congestion [34,141,142,190,195], especially when there are plenty of IoT services exchanging huge amounts of data continuously [24,25,187,190,196].

To avoid a single bottleneck, the distributed approach [133, 139–144,146] removes the central composite and distributes the load of data over multiple composites (see Fig. 11(b)). Although this improves load balancing, it introduces unnecessary network overhead as data passes through many mediators, even if data is unimportant for them, i.e., the more mediators, the more network overhead.

IoT services must exchange data as efficiently as possible (by minimising network hops), in order to avoid performance bottlenecks, achieve better response time and improve throughput [147]. A decentralised approach provides the most suitable data exchange for service composition since it requires only one network hop to pass data directly from a service producer to a service consumer (see Fig. 11(c)) [34,117,152,162,195,197–199]. For example, the data generated by *Booking.book* should be passed to *Display.showMap*, without passing through any other entity that does not require the produced data (e.g., the *CityManagement* composite or the *Vehicle* composite). In general, the amount of data transmitted in *SPark* may potentially be huge due to the large number of services involved of which sensors generate data continuously. Therefore, *SPark* requires a composition mechanism with support for decentralised data flows to provide an optimal QoS.

5.5. Separation of control, data and computation

Large-scale IoT systems may potentially span multiple administrative domains and exhibit a high degree of heterogeneity in many different forms [25,74,200]. For instance, there may be different service providers (e.g., Amazon AWS IoT and IBM Watson), a wide variety of programming languages (e.g., Swift and embedded C), multiple operating systems (e.g., Contiki and TinyOS) and different network communication protocols (e.g., CoAP and MQTT). For that reason, service composition mechanisms must provide the means to deal with such heterogeneity, in order to compose large-scale cross-domain IoT systems. This is, in fact, one of the major challenges for building IoT systems [200].

To enable flexible service composition in such heterogeneous environments, control flow, data flow and computation should be orthogonal [35,36,38,76,144,201,202]. This would enable separate reasoning of concerns so system developers can focus on IoT service composition, while service developers focus on efficient service functionality [37,74,76]. Consequently, services are workflow agnostic because workflow control flow is never embedded in the computation of many services. Moreover, the separation of control and computation facilitates workflow monitoring [29].

Fig. 12 illustrates a separation of control, data and computation. There are two computations defined in *A.opA*1 and *B.opB*1. The control flow part defines a workflow for the sequential invocation of *A.opA*1 and *B.opB*1, without considering data flow and computation. The data flow part defines intermediate processing of the output from *A.opA*1, before sending it to the input of *B.opB*1. When control is mixed with data, such processing represents an extra control flow step which is intrusive to the original workflow.

The separation of data flow will additionally enable separate data management for improving performance without considering control flow [35,162]. Similarly, control flow can be used

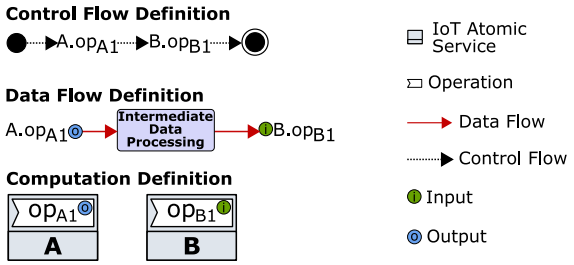

Fig. 12. Separation of control, data and computation.

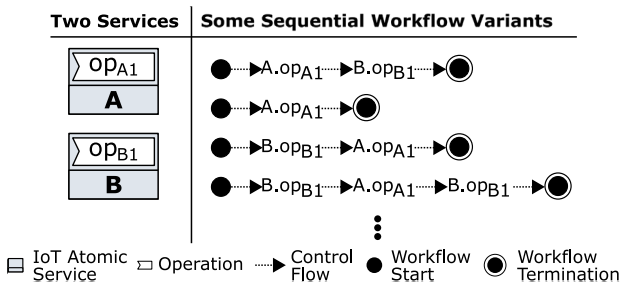

Fig. 13. Workflow variability.

in isolation for defining efficient deployment strategies that do not consider data flow. Moreover, different technologies can be used for implementing control flow, data flow and computation separately [35]. For instance, developers of *SPark* could use CoAP for passing control between services, the Blockchain for managing data flows and embedded C for implementing service computation. Providing independent reasoning of concerns also enables a separate validation and verification (V & V) of services.

When control is mixed with computation, a service provider and a service consumer are tightly coupled because invocations are originated in the consumer's computation. Thus, changing workflow control flow in one service requires further changes in the respective computation. This rigidity evidently limits the scalability of IoT systems because it is difficult to add, remove, change and replace services (and system workflow control flow). For that reason, reusing services is not possible at scale when control is mixed with computation [29,37,71,74,203].

Suppose there is a branching control flow structure in the computation of *Booking.book*, for invoking either *Provider1.payVisa* or *Provider2.payWallet*. As a result, *Booking.book* is not reusable as it is since it must be adapted to the requirements of other systems, e.g., it must be updated when two new payment providers come into play (e.g., *Provider3* and *Provider4*). As another example, SPark could need the parallel execution of *Payment.Pay* and *Vehicle.driveVehicle*, in order to improve concurrency. In this case, the sequence of invocations hardcoded in *SPark.findBestParking* must be replaced with a parallel control flow structure.

Generally speaking, large-scale IoT systems require separate reasoning of control, data and computation for independent maintenance, validation, verification, reuse and evolution of those concerns. This separation of concerns could also result in reduced time to market and reduced software production and maintenance costs [74].

5.6. Workflow variability

Large-scale IoT systems operate in highly dynamic environments subjected to variability caused by external or internal factors [39,41,204,205]. External factors are beyond the scope of the system and include changes in requirements and increasing workloads. Internal factors are associated with the system operation and include system failures and sub-optimal behaviours.

A service composition mechanism must support the definition of alternative behaviours, in order to adapt a composite to changes in both the external and the internal environment. As manually choosing alternative behaviours is a costly and inefficient management process, when there are many services in the composite, behaviours must be selected autonomously (i.e., with no human intervention) [206]. Thus, variability of behaviour is a crucial desideratum for the realisation of large-scale autonomous IoT systems [23,39–41,87,207].

Workflow variability [157,208] allows the definition of alternative control flow constructs (i.e., behaviours) in a service composition (see Fig. 13), and it is particularly useful for autonomously changing workflows at runtime. For instance, the *SensorNetwork* composite may define variable parallel behaviours for pulling data, depending on a vehicle's location. For one vehicle, it may pull data from *Sensor1* and *Sensor2*, whilst for another one it could pull data from *Sensor10*, *Sensor29* and *Sensor34*. As another example, consider multiple payment service providers that offer different QoS (which is always fluctuating according to different workloads). For this, a variable branching construct can be used to dynamically choose the *Provider* service with the best QoS.

When there is a family of related compositions, a *Greatest Common Denominator* [208] is a base variability model that can be reused for defining alternative compositions (also known as configurations). For example, suppose a SPark vehicle can be either *manual* or *self-driving*. A manual vehicle requires the services *GPS* and *Display*, whilst a self-driving one uses the service *DrivingCtr* in addition. For this, the *Vehicle* composite of Fig. 7 would be the *Greatest Common Denominator*, which can be reused for defining two alternative behaviours for two different vehicles.

Workflow variability is not only useful to accommodate the requirements of different vehicles, but also meaningful for different cities and users. Consider the case of Northern Ireland where many cities offer free on-street parking, and a few others (e.g., Belfast, Lisburn and Newry) impose a tariff [209]. Here, some *SPark* workflow variants may require a *Payment* composite, while others not. At the city scale, it would become very difficult to define workflows from scratch since many services would need to be changed and customised. Users can also require different workflows according to their needs, e.g., one user may require pre-payment, whereas another one may require post-payment. However, manually changing behaviour at runtime to accommodate different user requirements is infeasible. Thus, workflow variability represents a suitable solution for tackling the above scenarios.

In order to define workflow variability, a service composition mechanism must enable *total compositionality* by which all behaviours (e.g., operations) from the composed services are semantically available in a composite service [157,159]. This concept contrasts with *partial compositionality* where only named and selected behaviours are semantically preserved, leading to a combinatorial explosion of behaviours as the number of services increases [159].[4] Fig. 14 illustrates the dichotomy between total and partial compositionality. The lower section shows the preservation of operations $A.op_{A1}$ and $B.op_{B2}$ in the composite service, for the creation of a single workflow involving those operations. Remarkably, the upper section depicts a composite service that

[4] For example, there are $2^k - 1$ possible scenarios for pulling sensor data, where k is the number of sensors.

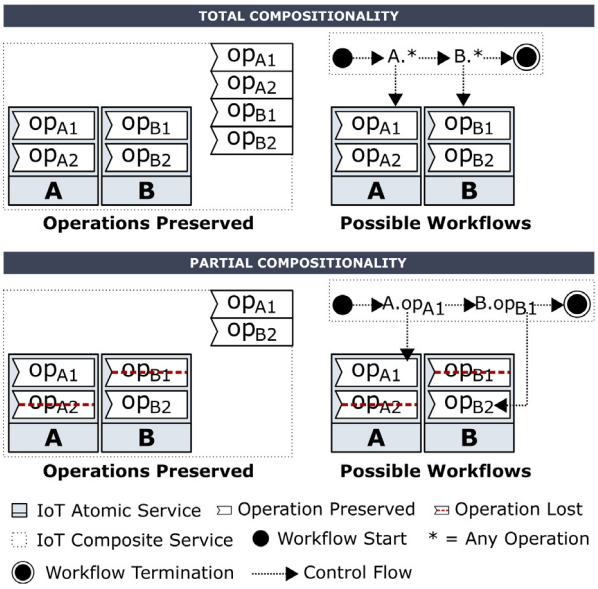

Fig. 14. Total compositionality vs partial compositionality.

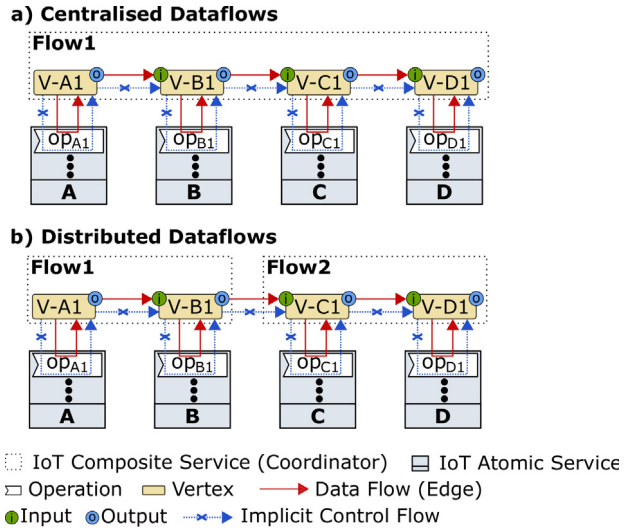

Fig. 15. Composition by dataflows.

preserves all the operations from the sub-services, which means that alternative workflows can be created. For instance, we could define a sequential workflow for the invocation of $B.op_{B1}$ and $A.op_{A2}$, or an alternative sequence for the execution of $A.op_{A1}$, $B.op_{B2}$ and $A.op_{A1}$. These behaviours are impossible to achieve with partial compositionality which allows the definition of only one fixed workflow at a time.

6. Analysis of IoT service composition mechanisms

This section reviews and analyses the fundamental semantics of current IoT service composition mechanisms, namely (centralised and distributed) dataflows, (centralised and distributed) orchestration, choreography and DX-MAN. The goal is to determine how well they fulfil the functional scalability requirements of IoT systems. For this analysis, we investigate the following research questions per mechanism:

- **RQ1:** Is there any architectural entity explicitly defining control flow?
- **RQ2:** Are atomic services location-agnostic of other atomic services?
- **RQ3:** Is it possible to distribute a workflow over multiple entities?
- **RQ4:** Do atomic services exchange data directly?
- **RQ5:** Do atomic services perform computation without passing any data or control to other atomic services?
- **RQ6:** Is there any notion of workflow variants?

For *R1–R4* and *R6*, we use a tick mark to indicate that a specific mechanism fulfils the requirement being analysed, or a cross mark to indicate the opposite. *RQ5* is the only requirement that uses a textual representation for showing which concerns (i.e., control, data and computation) are independent. To answer *RQ6*, we need to determine the compositionality of a mechanism which can be either total or partial (see Section 5.6). To do so, we investigate the following research questions:

- **RQ7:** What is the resulting type from composition?
- **RQ8:** How many workflows does the composition mechanism define?

6.1. Dataflows

Dataflows, or *Flow-Based Programming* [129–131], is a composition mechanism that defines a workflow using data transformations (e.g., filter, split, union and sort) as well as exogenous data exchange between services [71,132]. A dataflow description is a directed graph where vertices are asynchronous data processing units (invoking service operations), and edges are connections for passing data streams between vertices via the network (by message passing or events). A vertex explicitly defines input ports and output ports. When it receives data from all inputs, it performs some computation and writes results in output ports. The resulting data is then moved to other vertices via an edge. This process is illustrated in Fig. 15.

Dataflows can be centralised or distributed. A centralised dataflow [83,116] defines a single coordinator for managing an entire graph and exogenously invokes service operations. A distributed dataflow [37,38,133,134] partitions and distributes a complex graph over multiple coordinators that interact directly by exchanging data between vertices. Fig. 15(a) shows a centralised dataflow for a pipeline of services *A*, *B*, *C* and *D*. When the coordinator *Flow1* is triggered, vertex *V-A1* invokes $A.op_{A1}$ and passes the result to the vertex *V-B1* which, in turn, executes $B.op_{B1}$. Next, the result of *V-B1* is passed to the vertex *V-C1* which executes $C.op_{C1}$. Finally, the data of *V-C1* is moved to the vertex *V-D1* and then processed by $D.op_{D1}$. A distributed version of the same pipeline is shown in Fig. 15(b), where there is an edge between *V-B1* and *V-C1* for moving data from the *Flow1* composite to the *Flow2* composite.

Dataflows are increasingly popular for composing IoT systems. In particular, they are widely used for the Internet of Data (IoD) [210] which involves data collection from multiple sources (e.g., sensors), data analysis and control of the physical world. This paradigm has been referred to as Sense-Compute-Control (SCC) [211].[5] Currently, there are many platforms for composing IoT services using dataflows. Examples include Node-RED [83], COMPOSE [214], Glue.Things [116], LabVIEW [135], Paraimpu [215], Virtual Sensors [111], SpaceBrew [216], FogFlow

[5] Do not confuse data analysis tools [212,213] with dataflows. A data analysis tool is a software that allows the collection, storing, indexing, processing, monitoring and visualisation of data. On the other hand, dataflows specify how data is passed between services according to a dataflow graph specification.

Table 2
Compositionality of dataflows.

Composition mechanism	Composition unit	Resulting type from composition	Compositionality	Number of workflows
Centralised dataflows	IoT service	Workflow	Partial	1
Distributed dataflows	IoT service	Workflow	Partial	1

Table 3
Analysis of composition by dataflows w.r.t. the scalability desiderata.

	Centralised dataflows	Distributed dataflows
Explicit control flow	✗	✗
Service location transparency	✓	✓
Distributed workflows	✗	✓
Decentralised data flows	✗	✗
Separation of Control/Data/Computation	Data/Computation	Data/Computation
Workflow variability	✗	✗

[18], ASU VIPLE [217], ThingNet [126], Calvin [218], IoT Services Orchestration Layer [219], NoFlo [220] and many others [221–224].

As the Web 2.0 became more data-centric and user-friendly [65,71], dataflows have gained popularity for IoD through mashups. Mashups [225] are realised by dataflows [71,226], and they allow the composition and visualisation of data streams on a graphical user interface displayed on the Web [65,71,227]. Examples of mashup tools include WoTKit [228], IoTMaaS [196] and Clickscript [56].

Dataflows have been accepted as coordination languages since a graph is defined in a coordinator that exogenously invokes services according to a dataflow description [129,130,133]. A dataflow graph is typically created with a graphical editor and executed by an engine (i.e., the coordinator), and it is triggered by either timing constraints or events.

A dataflow coordinator is the only entity aware of service locations, and provides separation between data and computation. This separation allows service developers to focus on data stream computation while system developers wire up vertices exogenously [133]. However, despite the aforementioned advantages, decentralised data flows are not supported as data streams always pass through data flow coordinators.

Although passing data between vertices is explicitly defined in a dataflow graph, control flow is implicit in the collaborative data stream exchange. This is because control flow statements are not visible in the graph specification.

In general, a graph specification is a single flat workflow of named and selected service operations. It might seem that a distributed dataflow provides multiple workflows. However, this is not true because a distributed dataflow graph is just a single nested workflow, distributed over different coordinators. Thus, dataflows only provide partial compositionality and do not support workflow variability (see Table 2).

Table 3 summarises the results of our analysis of data flows w.r.t. the scalability desiderata. Both centralised dataflows and distributed dataflows have similar characteristics. The only difference is that the latter provides support for distributed workflows by partitioning a dataflow graph over multiple coordinators.

6.2. Orchestration

Orchestration can be centralised or distributed. Centralised orchestration [72,136–138] describes interactions between services from the perspective of a central coordinator (also known as orchestrator) which has control over all parties involved. It explicitly defines workflow control flow to coordinate the invocation of service operations, in order to realise some complex function that cannot be achieved by any individual service [69,70,229]. In a distributed orchestration, also known as "decentralised orchestration" [139–148], multiple coordinators collaboratively define workflow control flow.

An orchestration is typically defined using a workflow language such as BPEL [70,81,230–235] or BPMN [86,236]. The resulting workflow has tasks for passing control among services according to explicit control flow constructs (for sequencing, parallelising, branching and looping) [71]. In distributed orchestration, the interaction between coordinators can be done in three different ways: (i) *with an extra task* [140], (ii) *with two extra tasks* [141,142,148] or (iii) *without any extra task* [139,144,145]. In (i), an orchestration invokes the interface of another one using an external task to the system's workflow control flow. In (ii), there are two different tasks for receiving and passing control (and data) between two orchestrations. Finally, in (iii), two orchestrations interact by moving control (and data) directly between the tasks of the system's workflow control flow.

An orchestration engine is responsible for executing a workflow process by invoking service operations in a given order. Although traditional engines can be used (e.g., Camunda BPM workflow engine [86,237], Activiti [238,239] and AWS Step Functions [151]), recently we have seen the emergence of orchestration engines particularly designed for IoT systems (e.g., PROTEUS [149] and [150]).[6]

Fig. 16(a) illustrates a centralised orchestration for the services *A*, *B*, *C* and *D*, where the coordinator *Orch1* defines a "composite service" for the sequential invocation of $A.op_{A1}$, $B.op_{B1}$, $C.op_{C1}$ and $D.op_{D1}$. Three distributed versions are depicted in Fig. 16(b). In the former, *Orch1* defines a "composite service" for the sequential execution of $A.op_{A1}$ and $B.op_{B1}$, and then uses an extra task to pass control (and data) to *Orch2*. *Orch2* defines another "composite service" to sequentially invoke $C.op_{C1}$ and $D.op_{D1}$. The second distributed version uses two extra tasks for passing and receiving control (and data) between *Orch1* and *Orch2*. Finally, in the last distributed version, control and data are passed directly from $B.op_{B1}$ to $C.op_{C1}$.

A glance at Fig. 16 reveals that services are workflow agnostic because an orchestrator is the only entity aware of the location of other services. Having workflow agnostic services implies that an orchestrator provides separation between control flow and computation. This separation allows service developers to focus on efficient service functionality (i.e., computation) while system developers focus on workflow control flow in orchestrator(s).

Although a central coordinator facilitates the management of control flow logic, it easily becomes a performance bottleneck because all data passes through it [140,142]. This is because data

[6] A workflow engine can be deployed on either a specialised server [237] or a service bus like a Gateway [112,193,194].

Table 4
Compositionality of orchestration.

Composition mechanism	Composition unit	Resulting type from composition	Compositionality	Number of workflows
Centralised orchestration	IoT service	Workflow	Partial	1
Distributed orchestration	IoT service	Workflow	Partial	1

Table 5
Analysis of composition by orchestration w.r.t. the scalability desiderata.

	Centralised orchestration	Distributed orchestration
Explicit control flow	✓	✓
Service location transparency	✓	✓
Distributed workflows	✗	✓
Decentralised data flows	✗	✗
Separation of Control/Data/Computation	Control/Computation	Control/Computation
Workflow variability	✗	✗

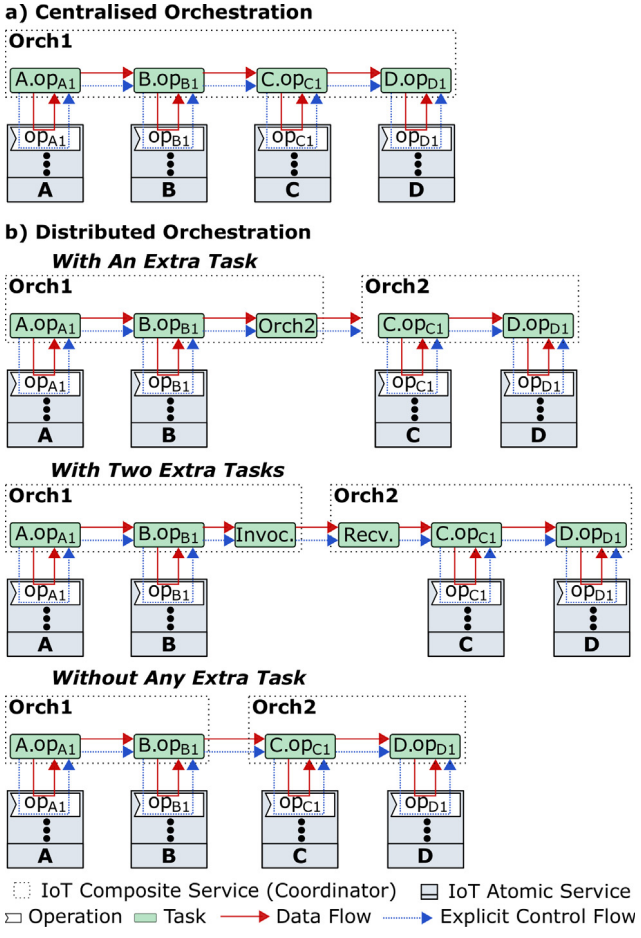

Fig. 16. Composition by orchestration.

follows control [71,162]. Furthermore, the resulting composite service is a single flat workflow, for the invocation of named and selected operations, which must be transformed into a service [159,240]. For that reason, orchestration only provides partial compositionality and does not support workflow variability (see Table 4).

Table 5 summarises the results of our analysis of orchestration w.r.t. the scalability desiderata, where we can see that centralised orchestration and distributed orchestration have similar characteristics. The only difference is that the latter supports distributed workflows via the partition of workflow control flow over multiple coordinators.

6.3. Choreography

A choreography describes service interactions from a global perspective using a public contract (also known as protocol) [136–138,152,153]. The contract specifies a "conversation" among participants via decentralised message exchanges, which can be modelled by a global observer using a choreography modelling language [122,138,153].[7] An interaction-based model allows the definition of event-driven or request–response messages by connecting required and provided interfaces. Examples include WS-CDL [241] and Let's Dance [242]. An interconnected interface model, on the other hand, allows the specification of control flow per participant. Examples include BPEL4Chor [240], Web Service Choreography Interface (WSCI) [243] and BPMN [244].[8]

A protocol defines roles for the collaborative realisation of a global workflow with no control over the internal details of the participants involved [229]. A role explicitly describes a participant workflow control flow in terms of expected and produced messages. When a concrete service instance plays a role, it must behave accordingly by exchanging messages with other instances, using either direct message passing (e.g., invoking REST APIs) or events [63,74,117,122].[9] This process is known as choreography enactment.

IoT is moving towards a more decentralised environment to reduce the bottleneck caused by centralised environments. As choreographies represent a natural way of dealing with such decentralisation, there are currently some platforms for composing IoT services by choreographies, e.g., CHOReVOLUTION [154], ChorSystem [246], Actorsphere [156], BeC^3 [117] and TraDE [35].

Fig. 17 illustrates a sequential choreography for the services A, B, C and D, where a protocol (defined with standard BPMN 2.0 notation [236]) specifies that $B.op_{B1}$ expects a message from $A.op_{A1}$, $C.op_{C1}$ a message from $B.op_{B1}$ and $D.op_{D1}$ a message from $C.op_{C1}$. When the choreography is enacted, there is a chain reaction that starts with the invocation of $A.op_{A1}$ and finishes with the execution of $D.op_{D1}$.

Fig. 17 shows that workflow control flow is explicitly defined in the protocol. During enactment, control is passed alongside data in every invocation, so services need to be aware of the location of other services. Of course, a service registry can be used, but this entails adding an "intrusive" element external to the composition (see Section 5.2).

[7] A Microservice architecture prefers choreography over orchestration to support decentralised workflows [122].

[8] The choice of the contract depends on the type of participants involved which can be either atomic services or orchestrations.

[9] Service participants are tightly coupled in terms of dependencies. In choreographies based on direct message-passing, services hardcode invocation calls in service computation. In event-driven choreographies, services are tightly coupled because senders and receivers agree a topic queue in advance [245].

Table 6
Compositionality of choreography.

Composition mechanism	Composition unit	Resulting type from composition	Compositionality	Number of workflows
Choreography	IoT service	Workflow	Partial	1

Table 7
Analysis of composition by choreography w.r.t. the scalability desiderata.

	Choreography
Explicit control flow	✓
Service location transparency	✗
Distributed workflows	✓
Decentralised data flows	✓
Separation of Control/Data/Computation	None
Workflow variability	✗

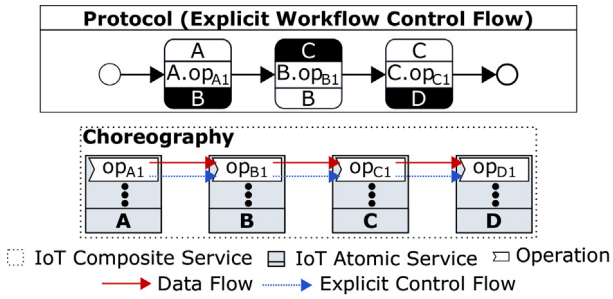

Fig. 17. Composition by choreography.

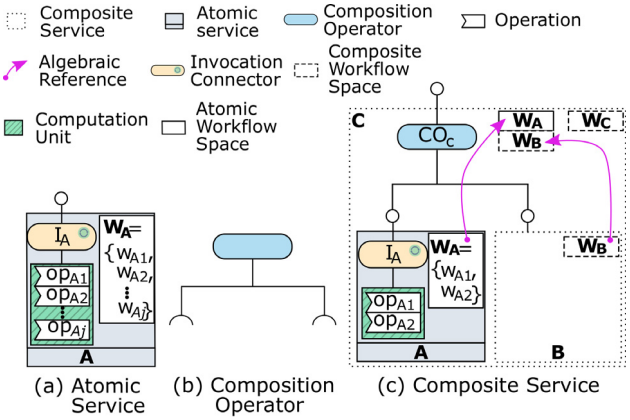

Fig. 18. DX-MAN model.

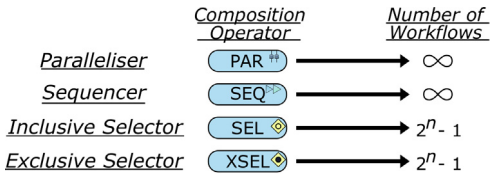

Fig. 19. DX-MAN composition operators.

Like orchestration, the resulting composition is a flat workflow for the invocation of selected and named operations (specified in the protocol). If the resulting workflow needs to be further composed, a choreography needs to be transformed into a service [159,240]. For that reason, a choreography is partially compositional and workflow variability is not supported (see Table 6). Table 7 summarises the results of our analysis of choreography w.r.t. the scalability desiderata.

6.4. DX-MAN

DX-MAN [157–159] is an algebraic model where IoT services and service composition operators are first-class semantic entities. A service is a stateless distributed unit of composition which can be either atomic or composite, and its semantics is a workflow space (i.e., a family of workflow variants). A *composition operator* defines variable control flows between families of workflow variants (see Fig. 18(b)).

An *atomic service* is formed by connecting an *invocation connector* with a *computation unit* (see Fig. 18(a)). It is a finite workflow space whose elements invoke a different operation implemented in the computation unit via an invocation connector. As a computation unit is semantically identical to a traditional SOA service (because it is a set of operations), DX-MAN atomic services and SOA atomic services are semantically different (cf. Section 2.1).

A *composite service* connects a composition operator with multiple (atomic or composite) sub-services (see Fig. 18(c)), which is equivalent to connecting a composition operator with multiple sub-workflow spaces. The result is a composite workflow space whose workflow variants invoke elements of atomic sub-workflow spaces and/or entire composite sub-workflow spaces, according to the control flow definition of the composition operator being used. There are composition operators for sequencing (i.e., *sequencer*), branching (i.e., *inclusive selector* and *exclusive selector*) and parallelism (i.e., *paralleliser*). Fig. 19 shows that the sequencer and paralleliser operators define infinite workflow variants, whilst the branching operators define $2^n - 1$ variants s.t. n is the total number of atomic sub-service operations plus the number of composite sub-workflow spaces. For further details on this matter, see [157].

In addition to composition operators, DX-MAN provides special transformation operators (called adapters) for sequencing, branching, parallelising, looping and guarding over individual workflow spaces. Hence, DX-MAN is Turing complete [247]. Currently, there is only one platform that implements the DX-MAN model [163], and it is available at https://github.com/damianarellanes/dxman.

In DX-MAN, a composition is done incrementally in a bottom-up fashion. So, a hierarchical connection structure of operators sits on top of atomic services. Fig. 20(a) shows a DX-MAN composition that involves four atomic services and three composite services. The first step is to model the atomic services A, B, C and D with the invocation connectors I_A, I_B, I_C and I_D. This process results in the atomic workflow spaces W_A, W_B, W_C and W_D, respectively. In the next hierarchy level, we create the composite services E and F. To do so, we use the composition operator SEQ_E to define a composite workflow space W_E from the atomic sub-workflow spaces W_A and W_B. Likewise, we define the composite workflow space W_F using the operator SEQ_F which operates on the atomic sub-workflow spaces W_C and W_D. Finally, we create the top-level composite service G. For this, we use the composition operator SEQ_G which defines the composite workflow space W_G from W_E and W_F.

Note that workflow spaces from sub-services are available in the next hierarchy level thanks to the algebraic semantics of the model. Consequently, the result of algebraic composition is not a single workflow, but a (potentially infinite) workflow space

Table 8
Compositionality of DX-MAN.

Composition mechanism	Composition unit	Resulting type from composition	Compositionality	Number of workflows
DX-MAN	IoT service	IoT service	Total	[1, ∞]

Table 9
Analysis of composition by DX-MAN w.r.t. the scalability desiderata.

	DX-MAN
Explicit control flow	✓
Service location transparency	✓
Distributed workflows	✓
Decentralised data flows	✓
Separation of Control/Data/Computation	Control/Data/ Computation
Workflow variability	✓

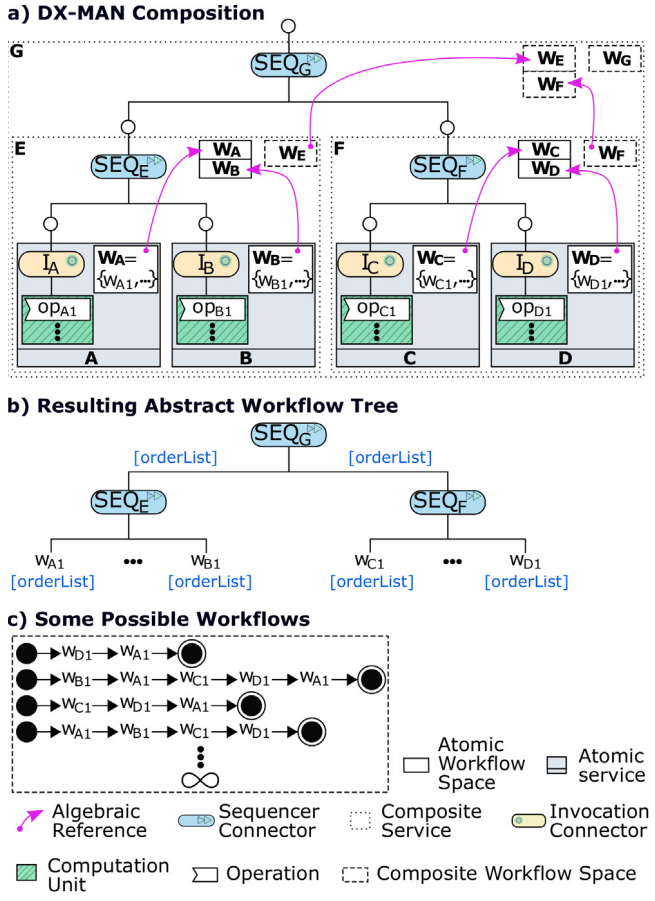

Fig. 20. Composition by DX-MAN.

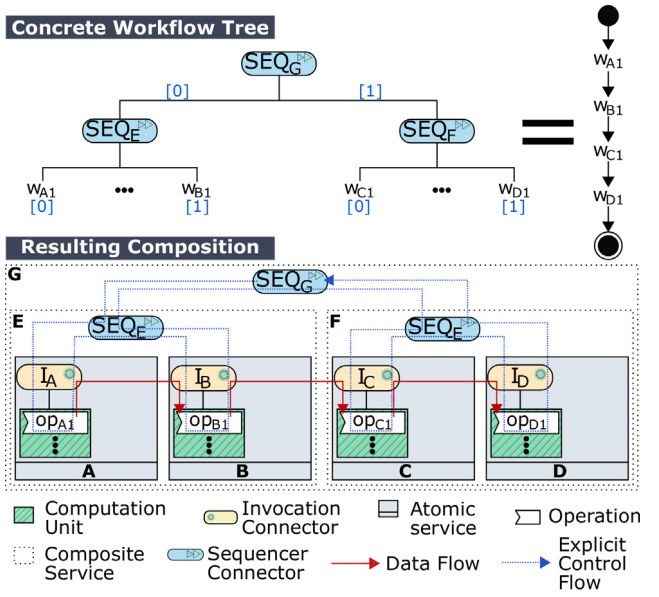

Fig. 21. A workflow variant derived from Fig. 20 for sequentially executing $A.op_{A1}$, $B.op_{B1}$, $C.op_{C1}$ and $D.op_{D1}$.

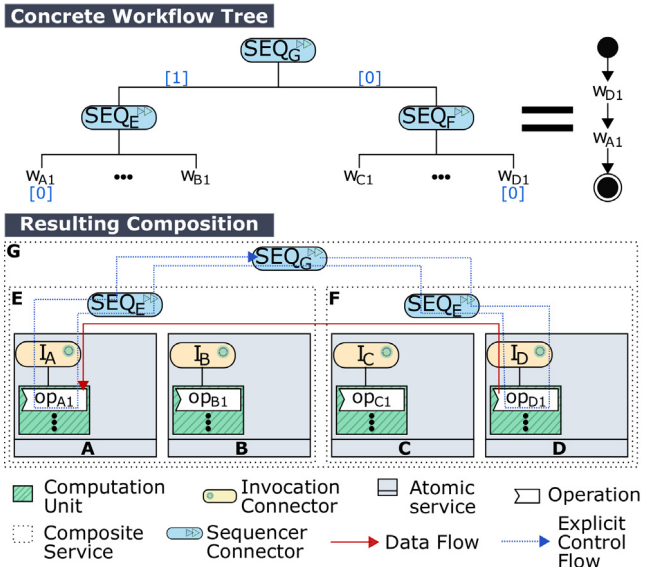

Fig. 22. A workflow variant derived from Fig. 20 for sequentially executing $D.op_{D1}$ and $A.op_{A1}$.

(i.e., a service) of workflow variants. Thus, DX-MAN supports total compositionality (see Table 8).

When a composition is done, an abstract workflow tree is automatically derived, which represents the hierarchical control flow structure of the composite (see Fig. 20(b)). To select a particular variant, a concrete workflow tree must be created, which is just a projection function over a composite workflow space. See [157,158] for further details.

Fig. 20(b) shows the abstract workflow tree of our example from which we can choose infinite sequential workflow variants (see Fig. 20(c)). One case is depicted in Fig. 21, where a concrete workflow tree is created for the sequential execution of $A.op_{A1}$, $B.op_{B1}$, $C.op_{C1}$ and $D.op_{D1}$. Fig. 22 shows another variant where a different concrete workflow tree is defined to invoke $A.op_{A1}$ and $D.op_{D1}$ sequentially.

Note that a DX-MAN composition has composition operators for the coordination of workflow control flow only. This is because data flow and control flow are modelled separately for each workflow variant. The paper [162] describes further details on how to define data flows per variant.

A glance at Figs. 21 and 22 reveals that data, control and computation are independent concerns. Thus, data is exchanged in a P2P fashion between atomic services, while composition operators coordinate workflow execution by passing control only. The coordination can be done in a distributed way since composition operators can be deployed on different things [159]. As

Table 10
Compositionality of IoT service composition mechanisms.

Composition mechanism	Composition unit	Resulting type from composition	Compositionality	Number of workflows
Centralised dataflows	IoT service	Workflow	Partial	1
Distributed dataflows	IoT service	Workflow	Partial	1
Centralised orchestration	IoT service	Workflow	Partial	1
Distributed orchestration	IoT service	Workflow	Partial	1
Choreography	IoT service	Workflow	Partial	1
DX-MAN	IoT service	IoT service	Total	$[1, \infty]$

Table 11
Analysis of scalability desiderata of IoT service composition mechanisms.

	Centralised dataflows	Distributed dataflows	Centralised orchestration	Distributed orchestration	Choreography	DX-MAN
Explicit control flow	✗	✗	✓	✓	✓	✓
Service location transparency	✓	✓	✓	✓	✗	✓
Distributed workflows	✗	✓	✗	✓	✓	✓
Decentralised data flows	✗	✗	✗	✗	✓	✓
Separation of Control/Data/Computation	Data/ Computation	Data/ Computation	Control/ Computation	Control/ Computation	None	Control/Data/ Computation
Workflow variability	✗	✗	✗	✗	✗	✓

Table 12
Satisfaction degree of IoT composition mechanisms w.r.t. scalability desiderata.

	b	c	r_b	r_c	s
Centralised dataflows	1	2	0.20	$0.\overline{66}$	27.78%
Distributed dataflows	2	2	0.40	$0.\overline{66}$	44.44%
Centralised orchestration	2	2	0.40	$0.\overline{66}$	44.44%
Distributed orchestration	3	2	0.60	$0.\overline{66}$	61.11%
Choreography	3	0	0.60	0.00	50.00%
DX-MAN	5	3	1.00	1.00	100.00%

coordination happens from outside services, computation units do not interact with one another, which results in independent distributed computations.

When an invocation connector receives control, it reads data from a decentralised data space (i.e., the Blockchain in the current implementation), invokes a service operation and writes results in the space. For that reason, only invocation connectors know the location of the connected computation units (i.e., service implementations). Furthermore, as invocation connectors perform operations on the data space, composition operators never exchange data during workflow execution. This (transparent) decentralised data exchange is achieved by the separation of control and data [162].

Table 9 summarises the results of our analysis of DX-MAN w.r.t. the scalability desiderata.

7. Evaluation

This section presents an evaluation of service composition mechanisms w.r.t. compositionality and the functional scalability requirements of IoT systems.

Table 10 summarises the analysis on compositionality presented in Section 6 to answer the research questions *RQ7* and *RQ8*. It shows that DX-MAN is the only mechanism that enables total compositionality since algebraic composition yields a service with a potentially infinite number of workflow variants. The other mechanisms define only one workflow at a time as the composition result is not a service, but a single flat workflow that invokes selected and named operations.

Research questions *RQ1-RQ6* have also been studied in Section 6. They enable us to analyse how well service composition mechanisms fulfil the scalability requirements: (i) explicit control flow; (ii) location transparency; (iii) distributed workflows; (iv) decentralised data flows; (v) separation of control, data and computation; and (vi) workflow variability.

Requirements (i), (ii), (iii), (iv) and (vi) are binary because they can be either supported (i.e., a tick mark) or not supported (i.e., a cross mark). Accordingly, we use Eq. (1) to determine the satisfaction degree of binary requirements for a specific composition mechanism,

$$r_b(b) = \frac{b}{5} \tag{1}$$

where r_b is the satisfaction degree of binary requirements and b is the number of supported binary requirements by the mechanism s.t. $b \in (\mathbb{N} \cap [0, 5])$ and $r_b \in (\mathbb{Q} \cap [0, 1])$.

The requirement (v) is quinary since it admits $2^3 - 3$ possible results: *None*, *Control/Data*, *Control/Computation*, *Data/Computation* and *Control/Data/Computation*. This is because (v) considers three different concerns (i.e., control, data and computation) and discards options involving only one concern. When (v) is *None*, the requirement support becomes zero. Accordingly, Eq. (2) determines the degree of separation of concerns for a specific composition mechanism,

$$r_c(c) = \frac{c}{3} \tag{2}$$

where r_c is the degree of separation of concerns and c is the number of independent concerns supported by the mechanism s.t. $c \in (\mathbb{N} \cap \{0, 2, 3\})$ and $r_c \in (\mathbb{Q} \cap [0, 1])$.

To determine the satisfaction degree of a composition mechanism w.r.t. scalability desiderata, Eqs. (1) and (2) are used in Eq. (3). The result is a percentage that takes into account five binary requirements and one quinary requirement. A higher percentage means a higher satisfaction.

$$s(r_b, r_c) = (r_b \times \frac{5}{6} + r_c \times \frac{1}{6}) \times 100 \tag{3}$$

where s is the overall satisfaction degree w.r.t. all scalability requirements s.t. $s \in [0, 100]$.

Table 11 summarises our analysis of scalability desiderata and Table 12 shows the respective satisfaction degrees. Our interpretation of the results is presented below.

Centralised dataflows is the worst mechanism because it supports only one binary requirement (i.e., location transparency) and separates two concerns (i.e., data and computation). This means that the satisfaction degree of binary requirements is $0.\overline{20}$, while the satisfaction degree of separation of concerns is $0.\overline{66}$. Thus, the overall satisfaction degree of centralised dataflows is 27.78%. Distributed dataflows provide distributed workflows in addition. Consequently, it possesses a (higher) satisfaction degree

of binary requirements equal to 0.40 and, therefore, a (higher) overall satisfaction degree of 44.44%.

Although centralised orchestration has the same satisfaction degree as distributed dataflows, it supports different binary requirements (i.e., explicit control flow and location transparency) and separates different concerns (i.e., control and computation). Distributed orchestration is similar, yet different. It offers distributed workflows in addition for a (higher) satisfaction of binary requirements of 0.60 and, therefore, a (higher) overall satisfaction degree of 61.11%.

Choreography covers three binary requirements (i.e., explicit control flow, distributed workflows and decentralised data flows) and does not provide any separation of concerns since control and data are mixed in service computation. This means that the satisfaction degree of binary requirements is 0.60, with a null separation of concerns. Overall, choreography fulfils scalability requirements to a degree of 50%.

DX-MAN is the only mechanism that fulfils all binary requirements and provides the separation of control, data and computation. It is also the only one that supports workflow variability because of total compositionality (see Table 10). Accordingly, the satisfaction degrees of binary requirements and the separation of concerns are both 1. Thus, DX-MAN best fulfils the desiderata with a satisfaction degree of 100%.

8. Discussion

This section discusses the results presented in Section 7 and covers additional issues concerning compositionality, scalability requirements and the relationship between them.

Our results show that explicit control flow is supported by the majority of the mechanisms. In particular, orchestration defines control flow in orchestrators, choreography in a protocol and DX-MAN in composition operators. Dataflows is the only one that does no support such a requirement.

Workflow distribution is also meet by the majority of the mechanisms, except centralised dataflows and centralised orchestration where a coordinator fully governs a workflow.

Almost all composition mechanisms use a coordinator to exogenously define workflow(s) and, therefore, ensuring the separation of at least two concerns. In particular, orchestration and DX-MAN separate control and computation, while dataflows orthogonalises data and computation. Such separation enables location transparency as only coordinators are aware of atomic service locations. Choreography is the only mechanism without any support for location transparency, since control and data are mixed with computation.

There is a special choreography implementation based on the data-driven paradigm in which some computation is triggered once input data becomes available [199]. As a protocol must explicitly define workflow control flow, the analysis presented in Section 6 is also applicable.

Generally speaking, avoiding coordinators allows choreography styles to support decentralised data flows. So, there is a trade-off between coordination and decentralised data flows. DX-MAN obviates this problem by separating data in addition to control and computation. This separation enables the realisation of decentralised data flows, without considering neither control nor computation. So, data never passes through composition operators (i.e., coordinators) [162].

Some orchestration approaches [34,146,195,198] partially separate control from data algorithmically. To achieve this, coordinators pass data references alongside control, rather than exchanging data values. However, analysing data dependencies to extract references is a challenging task because data and control are still semantically entangled.

For orchestration and dataflows, a coordinator is commonly referred to as a composite service. However, it is just a composition of specific operations (i.e., a workflow) that must be transformed into a service, rather than a composition of entire services (with all their provided operations). Only DX-MAN achieves an actual composition of services (not operations) as it defines a composite service without any transformation step while preserving all service operations from which multiple workflow variants can be derived. Thus, algebraic composition equates to total compositionality which, in turn, implies workflow variability. A DX-MAN composite service is in fact semantically equivalent to a (potentially infinite) set of orchestrations. This is because each element in a composite workflow space is a different composition workflow as regarded by existing mechanisms. For any existing composition mechanism to be equivalent to a composite workflow space, it would require to include all possible combinations of execution paths in the workflow definition, leading to a combinatorial explosion problem. While it is true a configurable workflow [208] can deal with this issue, it does not define multiple workflows at a time but just a single workflow that can be manually configured.

In this paper, we analyse the fundamental semantics of current service composition mechanisms that allow the definition of IoT workflows. However, other mechanisms must be considered even if they do not allow the explicit definition of behaviour, e.g., port connections [248] and ensemble-based composition [249]. Moreover, we did not analyse service interactions where there is no definition of composite services such as direct message passing [250], broker-based interactions [112] and event-driven interactions [251]. For this, please refer to [74].

Different composition mechanisms can coexist in the same system. For example, it is possible to build a system where some services are composed using orchestration while others are composed by choreography [155,252]. Our intention is not to analyse the combination of different composition mechanisms, but to analyse them individually. Clearly, our analysis results apply to such a combination.

9. Conclusions

Functional scalability becomes a crucial concern for IoT systems as the number of available services increases. For that reason, we need to look back at the foundations of service composition in order to tackle this challenging problem. In this paper, we analysed the semantics of current IoT service composition mechanisms, using an evaluation framework that considers six functional scalability requirements: (i) explicit control flow; (ii) distributed workflows; (iii) location transparency; (iv) decentralised data flows; (v) separation of control, data and computation; and (vi) workflow variability. Our results suggest that DX-MAN is the composition mechanism that best fulfils the functional scalability requirements of IoT systems. This is not surprising since such a model was designed with scalability desiderata in mind.

It is important to note that, except DX-MAN, there are no new composition mechanisms developed in the last decade. This crisis is a worrying situation that must be prioritised and remedied because of the inherent scale that IoT systems pose. Only DX-MAN was designed with this in mind, but we expect further developments on this in the coming years.

We do not claim that the scalability requirements we analyse here are complete, as other characteristics must also be considered. Nevertheless, we believe our evaluation framework has included the most critical ones and provides a useful starting point for further extensions that consider other functional scalability desiderata such as dynamic reconfiguration, the number of messages exchanged and the support for implicit/explicit data flows. The framework can thus be refined by other researchers when conducting further studies on IoT service composition mechanisms.

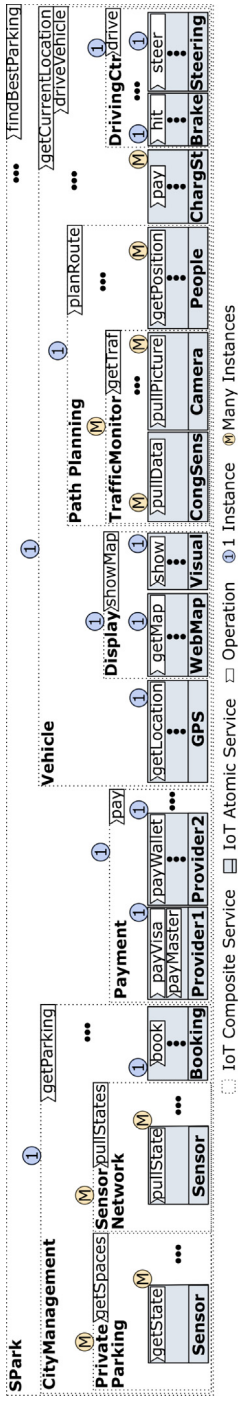

Fig. A.1. A larger view of *SPark* services.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The first author would like to acknowledge the financial support provided by the National Council of Science and Technology (CONACyT) under the grant 280863/411891.

Appendix A

A larger view of *SPark* is shown in Fig. A.1, where there are new composite services representing *Private Parkings* of a smart city. Also, the *Display* service has been converted into a composite that uses the atomic services *WebMap* and *Visual*. There is also a new composite service called *PathPlanning* that composes multiple *TrafficMonitor* composites and multiple atomic *People* services. In particular, a *TrafficMonitor* composite has multiple atomic services that represent congestion sensors and cameras (distributed across a city). Furthermore, the *Vehicle* composite has been endowed with multiple atomic *ChargSt* services (for charging the battery of a self-driving vehicle). Finally, the *DrivingCtr* service has been converted into a composite service with the atomic services *Brake* and *Steering*.

It is important to note that the larger view shown in Fig. A.1 can be expanded into an even larger composition which may become infeasible to depict in a single document.

Appendix B

Fig. B.1 illustrates the complete workflow of *SPark* which starts with the on-demand execution of *SPark.findBestParking*. Next, the *SPark* composite gets the current vehicle's location by invoking the operation *Vehicle.getCurrentLocation* which, in turn, invokes *GPS.getLocation*. Then, it invokes the *CityManagement.getParking* operation which internally executes the *SensorNetwork.pullStates* operation from the nearest InfoStation. In particular, *SensorNetwork.pullStates* pulls the states (in parallel) from the sensors close to the vehicle's location, using the respective *Sensor.pullState* operations. Next, the *CityManagement.getParking* operation uses the sensor states to determine the best (i.e., the free and nearest) parking space and then reserves that parking space using *Booking.book*. Then, control is passed to the *Payment.pay* operation which decides to invoke *Provider1.payVisa*, *Provider1.payMastercard* or *Provider2.payWallet*. Finally, the *Vehicle.driveVehicle* is invoked for implicitly calling *Display.showMap* and *DrivingCtr.drive*, in that order.

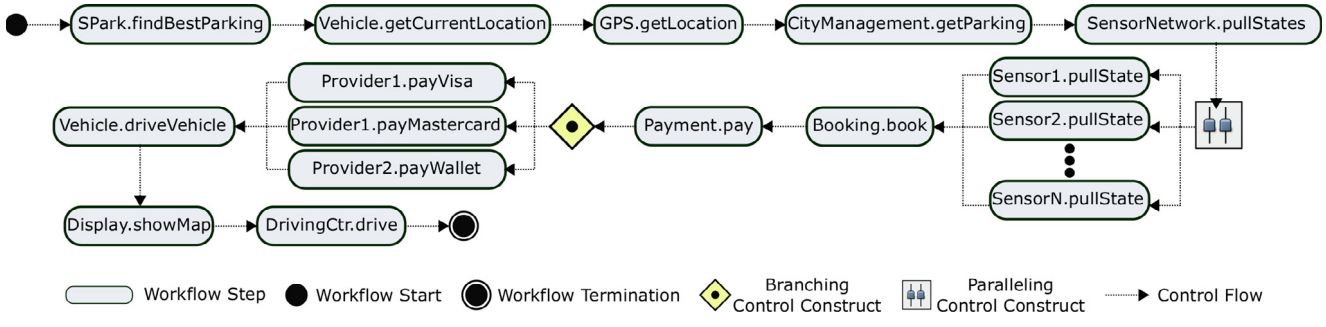

Fig. B.1. *SPark* workflow control flow.

References

[1] IoT Analytics, State of the IoT 2018: Number of IoT devices now at 7B – market accelerating, 2018, URL https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/.

[2] Chayan Sarkar, Akshay Uttama Nambi S. N, R. Venkatesha Prasad, Abdur Rahim, Ricardo Neisse, Gianmarco Baldini, DIAT: A scalable distributed architecture for IoT, IEEE Internet Things J. 2 (3) (2015) 230–239.

[3] Marzieh Hamzei, Nima Jafari Navimipour, Toward efficient service composition techniques in the internet of things, IEEE Internet Things J. 5 (5) (2018) 3774–3787.

[4] Asrin Vakili, Nima Jafari Navimipour, Comprehensive and systematic review of the service composition mechanisms in the cloud environments, J. Netw. Comput. Appl. 81 (2017) 24–36.

[5] Parvaneh Asghari, Amir Masoud Rahmani, Hamid Haj Seyyed Javadi, Service composition approaches in IoT: A systematic review, J. Netw. Comput. Appl. 120 (2018) 61–77.

[6] Steen Maarten Van, Andrew S. Tanenbaum, Distributed Systems, third ed., CreateSpace, London, UK, 2017.

[7] Xiang Sun, Nirwan Ansari, EdgeIoT: Mobile edge computing for the internet of things, IEEE Commun. Mag. 54 (12) (2016) 22–29.

[8] Ling Li, Shancang Li, Shanshan Zhao, QoS-aware scheduling of services-oriented internet of things, IEEE Trans. Ind. Inf. 10 (2) (2014) 1497–1505.

[9] Stelios Sotiriadis, Nik Bessis, Cristiana Amza, Rajkumar Buyya, Elastic load balancing for dynamic virtual machine reconfiguration based on vertical and horizontal scaling, IEEE Trans. Serv. Comput. 12 (2) (2019) 319–334.

[10] Jesús Alejandro Cárdenes Cabré, Doina Precup, Ricardo Sanz, Horizontal and vertical self-adaptive cloud controller with reward optimization for resource allocation, in: International Conference on Cloud and Autonomic Computing, ICCAC, IEEE, 2017, pp. 184–185.

[11] T. Ramalingeswara Rao, Pabitra Mitra, Ravindara Bhatt, A. Goswami, The big data system, components, tools, and technologies: a survey, Knowl. Inf. Syst. (2018) 1–81, Advance online publication.

[12] Chii Chang, Satish Narayana Srirama, Rajkumar Buyya, Internet of things (IoT) and new computing paradigms, in: Rajkumar Buyya, Satish Narayana Srirama (Eds.), Fog and Edge Computing, Wiley Publishing, Hoboken, NJ, USA, 2019, pp. 3–23.

[13] Jian Wu, Qianhui Liang, Elisa Bertino, Improving scalability of software cloud for composite web services, in: International Conference on Cloud Computing, CLOUD, IEEE, 2009, pp. 143–146.

[14] Radu Calinescu, Lars Grunske, Marta Kwiatkowska, Raffaela Mirandola, Giordano Tamburrelli, Dynamic QoS management and optimization in service-based systems, IEEE Trans. Softw. Eng. 37 (3) (2011) 387–409.

[15] Altti Ilari Maarala, Xiang Su, Jukka Riekki, Semantic reasoning for context-aware internet of things applications, IEEE Internet Things J. 4 (2) (2017) 461–473.

[16] Yi Xu, Abdelsalam Helal, Scalable cloud–sensor architecture for the internet of things, IEEE Internet Things J. 3 (3) (2016) 285–298.

[17] Roberto Girau, Salvatore Martis, Luigi Atzori, Lysis: A platform for IoT distributed applications over socially connected objects, IEEE Internet Things J. 4 (1) (2017) 40–51.

[18] Bin Cheng, Gürkan Solmaz, Flavio Cirillo, Ernö Kovacs, Kazuyuki Terasawa, Atsushi Kitazawa, FogFlow: Easy programming of IoT services over cloud and edges for smart cities, IEEE Internet Things J. 5 (2) (2018) 696–707.

[19] Emanuel Ferreira Coutinho, Flávio Rubens de Carvalho Sousa, Paulo Antonio Leal Rego, Danielo Gonçalves Gomes, José Neuman de Souza, Elasticity in cloud computing: a survey, Ann. Telecommun. 70 (7) (2015) 289–309.

[20] Rajkumar Buyya, Amir Dastjerdi, Internet of Things: Principles and Paradigms, Morgan Kaufmann, Cambridge, MA, USA, 2016.

[21] David G. Messerschmitt, Clemens Szyperski, Software Ecosystem: Understanding an Indispensable Technology and Industry, first ed., MIT Press, Cambridge, MA, USA, 2003.

[22] Roy Want, Bill N. Schilit, Scott Jenson, Enabling the internet of things, Computer 48 (1) (2015) 28–35.

[23] Peter Feiler, Richard P. Gabriel, John Goodenough, Rick Linger, Tom Longstaff, Rick Kazman, Mark Klein, Linda Northrop, Douglas Schmidt, Kevin Sullivan, Kurt Wallnau, Ultra-Large-Scale Systems the Software Challenge of the Future, Technical Report, Software Engineering Institute, Carnegie Mellon University, 2006.

[24] Luigi Atzori, Antonio Iera, Giacomo Morabito, The internet of things: A survey, Comput. Netw. 54 (15) (2010) 2787–2805.

[25] Flávia C. Delicato, Paulo F. Pires, Thais Batista, The resource management challenge in IoT, in: Flávia C. Delicato, Paulo F. Pires, Thais Batista (Eds.), Resource Management for Internet of Things, Springer, Cham, Switzerland, 2017, pp. 7–18.

[26] Damian Roca, Daniel Nemirovsky, Mario Nemirovsky, Rodolfo Milito, Mateo Valero, Emergent behaviors in the internet of things: The ultimate ultra-large-scale system, IEEE Micro 36 (6) (2016) 36–44.

[27] Hermann Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications, second ed., Springer, New York, NY, USA, 2011.

[28] Martin Törngren, Paul T. Grogan, How to deal with the complexity of future cyber-physical systems? Designs 2 (4) (2018) 1–16.

[29] Netflix, Conductor, 2019, URL https://netflix.github.io/conductor/.

[30] Martin Fowler, What do you mean by "event-driven"? 2017, URL https://martinfowler.com/articles/201701-event-driven.html.

[31] Ian Sommerville, Dave Cliff, Radu Calinescu, Justin Keen, Tim Kelly, Marta Kwiatkowska, John Mcdermid, Richard Paige, Large-scale complex IT systems, Commun. ACM 55 (7) (2012) 71–77.

[32] Reza Rezaei, Thiam Kian Chiew, Sai Peck Lee, An interoperability model for ultra large scale systems, Adv. Eng. Softw. 67 (2014) 22–46.

[33] Valérie Issarny, Nikolaos Georgantas, Sara Hachem, Apostolos Zarras, Panos Vassiliadist, Marco Autili, Marco Aurélio Gerosa, Amira Ben Hamida, Service-oriented middleware for the future internet: state of the art and research directions, J. Internet Serv. Appl. 2 (1) (2011) 23–45.

[34] Adam Barker, Jon B. Weissman, Jano I. Van Hemert, Reducing data transfer in service-oriented architectures: The circulate approach, IEEE Trans. Serv. Comput. 5 (3) (2012) 437–449.

[35] Michael Hahn, Uwe Breitenbücher, Oliver Kopp, Frank Leymann, Modeling and execution of data-aware choreographies: an overview, Compu. Sci. - Res. Dev. 33 (3) (2018) 329–340.

[36] Gio Wiederhold, Peter Wegner, Stefano Ceri, Towards megaprogramming: A paradigm for component-based programming, Commun. ACM 35 (11) (1992) 89–99.

[37] Nam Ky Giang, Rodger Lea, Victor C.M. Leung, Exogenous coordination for building fog-based cyber physical social computing and networking systems, IEEE Access 6 (2018) 31740–31749.

[38] Nam Ky Giang, Rodger Lea, Victor C.M. Leung, Developing applications in large scale, dynamic fog computing: A case study, Softw. - Pract. Exp. (2019) 1–14.

[39] Matthias Galster, Uwe Zdun, Danny Weyns, Rick Rabiser, Bo Zhang, Michael Goedicke, Gilles Perrouin, Variability and complexity in software design: Towards a research agenda, SIGSOFT Softw. Eng. Notes 41 (6) (2017) 27–30.

[40] Danny Weyns, Gowri Sankar Ramachandran, Ritesh Kumar Singh, Self-managing internet of things, in: A Min Tjoa, Ladjel Bellatreche, Stefan Biffl, Jan van Leeuwen, Jiří Wiedermann (Eds.), SOFSEM 2018: Theory and Practice of Computer Science, in: Lecture Notes in Computer Science, vol. 10706, Springer, Cham, Switzerland, 2018, pp. 67–84.

[41] Gerald Holl, Paul Grünbacher, Rick Rabiser, A systematic review and an expert survey on capabilities supporting multi product lines, Inf. Softw. Technol. 54 (8) (2012) 828–852.

[42] Kevin Ashton, That 'internet of things' thing, RFID J. (2009) 1.

[43] Mark Weiser, The computer for the 21st century, Sci. Am. 265 (3) (1991) 94–105.

[44] ITU-T, Overview of the Internet of things, Technical Report ITU-T Y.4000/Y.2060, International Telecommunication Union, 2012.

[45] Karl Popper, Three worlds, The Tanner Lecture on Human Values, The University of Michigan, 1978.

[46] Ing-Ray Chen, Jia Guo, Fenye Bao, Trust management for SOA-based IoT and its application to service composition, IEEE Trans. Serv. Comput. 9 (3) (2016) 482–495.

[47] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, Frank Leymann, Service-oriented computing: State of the art and research challenges, Computer 40 (11) (2007) 38–45.

[48] Sherif Abdelwahab, Bechir Hamdaoui, Mohsen Guizani, Taieb Znati, Cloud of things for sensing-as-a-service: Architecture, algorithms, and use case, IEEE Internet Things J. 3 (6) (2016) 1099–1112.

[49] Robert Brzoza-Woch, Piotr Nawrocki, FPGA-based web services – infinite potential or a road to nowhere? IEEE Internet Comput. 20 (1) (2016) 44–51.

[50] Michele Nitti, Virginia Pilloni, Giuseppe Colistra, Luigi Atzori, The virtual object as a major element of the internet of things: A survey, IEEE Commun. Surv. Tutor. 18 (2) (2016) 1228–1240.

[51] Li Da Xu, Wu He, Shancang Li, Internet of things in industries: A survey, IEEE Trans. Ind. Inf. 10 (4) (2014) 2233–2243.

[52] Dominique Guinard, Vlad Trifa, Stamatis Karnouskos, Patrik Spiess, Domnic Savio, Interacting with the SOA-based internet of things: Discovery, query, selection, and on-demand provisioning of web services, IEEE Trans. Serv. Comput. 3 (3) (2010) 223–235.

[53] Karl D. Gottschalk, Stephen Graham, Heather Kreger, James Snell, Introduction to Web services architecture, IBM Syst. J. 41 (2) (2002) 170–177.

[54] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, Sanjiva Weerawarana, Unraveling the web services web: An introduction to SOAP, WSDL, and UDDI, IEEE Internet Comput. 6 (2) (2002) 86–93.

[55] Nissanka B. Priyantha, Aman Kansal, Michel Goraczko, Feng Zhao, Tiny web services: Design and implementation of interoperable and evolvable sensor networks, in: ACM Conference on Embedded Network Sensor Systems, SenSys, ACM, 2008, pp. 253–266.
[56] Dominique Guinard, Vlad Trifa, Friedemann Mattern, Erik Wilde, From the internet of things to the web of things: resource-oriented architecture and best practices, in: Dieter Uckelmann, Mark Harrison, Florian Michahelles (Eds.), Architecting the Internet of Things, Springer, Berlin, Germany, 2011, pp. 97–129.
[57] Sylvain Cherrier, Yacine M. Ghamri-Doudane, The "object-as-a-service" paradigm, in: Global Information Infrastructure and Networking Symposium, GIIS, IEEE, 2014, pp. 1–7.
[58] Douglas K. Barry, David Dick, Web Services, Service-Oriented Architectures, and Cloud Computing: The Savy Manager's Guide, second ed., Morgan Kaufmann, Burlington, MA, USA, 2013.
[59] Kung-Kiu Lau, Simone Di Cola, An Introduction to Component-based Software Development, first ed., World Scientific, Singapore, 2017.
[60] Michael Bell, SOA Modeling Patterns for Service Oriented Discovery and Analysis, first ed., Wiley Publishing, Hoboken, NJ, USA, 2010.
[61] Chandrashekar Jatoth, G.R. Gangadharan, Rajkumar Buyya, Computational intelligence based QoS-aware web service composition: A systematic literature review, IEEE Trans. Serv. Comput. 10 (3) (2017) 475–492.
[62] Dominique Guinard, Towards the web of things: Web mashups for embedded devices, in: International World Wide Web Conference, WWW, ACM, 2009, pp. 1–8.
[63] Martin Bauer, Nicola Bui, Jourik De Loof, Carsten Magerkurth, Andreas Nettsträter, Julinda Stefa, Joachim W. Walewski, IoT reference model, in: Alessandro Bassi, Martin Bauer, Martin Fiedler, Thorsten Kramp, Rob van Kranenburg, Sebastian Lange, Stefan Meissner (Eds.), Enabling Things to Talk: Designing IoT Solutions with the IoT Architectural Reference Model, Springer, Berlin, Germany, 2013, pp. 113–162.
[64] João Rufino, Muhammad Alam, Joaquim Ferreira, Abdur Rehman, Kim Fung Tsang, Orchestration of containerized microservices for IIoT using docker, in: International Conference on Industrial Technology, ICIT, IEEE, 2017, pp. 1532–1536.
[65] Cesare Pautasso, RESTful web service composition with BPEL for REST, Data Knowl. Eng. 68 (9) (2009) 851–866.
[66] Roy Fielding, Architectural Styles and the Design of Network-based Software Architectures (Ph.D. thesis), University of California, Irvine, 2000.
[67] Cesare Pautasso, Olaf Zimmermann, Frank Leymann, Restful web services vs. "Big'" web services: Making the right architectural decision, in: International Conference on World Wide Web, WWW, ACM, 2008, pp. 805–814.
[68] Dominique Guinard, Iulia Ion, Simon Mayer, In search of an internet of things service architecture: REST or WS-*? a developers' perspective, in: Alessandro Puiatti, Tao Gu (Eds.), Mobile and Ubiquitous Systems: Computing, Networking, and Services, in: Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 104, Springer, Berlin, Germany, 2012, pp. 326–337.
[69] Choonhwa Lee, Chengyang Wang, Eunsam Kim, Sumi Helal, Blueprint flow: A declarative service composition framework for cloud applications, IEEE Access 5 (2017) 17634–17643.
[70] Rebeca P. Diaz Redondo, Ana Fernandez Vilas, Manuel Ramos Cabrer, Jose J. Pazos Arias, Marta Rey Lopez, Enhancing residential gateways: OSGi service composition, IEEE Trans. Consum. Electron. 53 (1) (2007) 87–95.
[71] Hye-Young Paik, Angel Lagares Lemos, Moshe Chai Barukh, Boualem Benatallah, Aarthi Natarajan, Web Service Implementation and Composition Techniques, first ed., Springer, Cham, Switzerland, 2017.
[72] Angel Lagares Lemos, Florian Daniel, Boualem Benatallah, Web service composition: A survey of techniques and tools, ACM Comput. Surv. 48 (3) (2015) 1–41.
[73] Remco Dijkman, Marlon Dumas, Service-oriented design: a multi-viewpoint approach, Int. J. Coop. Inf. Syst. 13 (4) (2004) 337–368.
[74] Damian Arellanes, Kung-Kiu Lau, Analysis and classification of service interactions for the scalability of the internet of things, in: International Congress on Internet of Things, ICIOT, IEEE, 2018, pp. 80–87.
[75] Matthew Shields, Control- versus data-driven workflows, in: Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, Matthew Shields (Eds.), Workflows for E-Science: Scientific Workflows for Grids, Springer, London, UK, 2007, pp. 167–173.
[76] Amazon, Amazon simple workflow service (SWF), 2019, URL https://aws.amazon.com/swf/.
[77] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, Yang Zhao, Scientific workflow management and the Kepler system, Concurr. Comput.: Pract. Exper. 18 (10) (2006) 1039–1065.
[78] Carl Adam Petri, Kommunikation Mit Automaten (Ph.D. thesis), Institut für Instrumentelle Mathematik, 1962.
[79] Yong Zhao, Michael Wilde, Ian Foster, Virtual data language: A typed workflow notation for diversely structured scientific data, in: Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, Matthew Shields (Eds.), Workflows for E-Science: Scientific Workflows for Grids, Springer, London, UK, 2007, pp. 258–275.
[80] Tom Goodale, Gabrielle Allen, Gerd Lanfermann, Joan Massó, Thomas Radke, Edward Seidel, John Shalf, The cactus framework and toolkit: Design and applications, in: José M.L.M. Palma, A. Augusto Sousa, Jack Dongarra, Vicente Hernández (Eds.), High Performance Computing for Computational Science — VECPAR 2002, in: Lecture Notes in Computer Science, vol. 2565, Springer, Berlin, Germany, 2003, pp. 197–227.
[81] OASIS, Web services business process execution language version 2.0, 2007, URL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.
[82] Ian Taylor, Matthew Shields, Ian Wang, Andrew Harrison, The triana workflow environment: Architecture and applications, in: Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, Matthew Shields (Eds.), Workflows for E-Science: Scientific Workflows for Grids, Springer, London, UK, 2007, pp. 320–339.
[83] OpenJS Foundation, Node-RED: Flow-based programming for the internet of things, 2019, URL https://nodered.org/.
[84] Li Da Xu, Wattana Viriyasitavat, A novel architecture for requirement-oriented participation decision in service workflows, IEEE Trans. Ind. Inf. 10 (2) (2014) 1478–1485.
[85] Nathaniel Palmer, Swenson Keith, Reddy Surendra, Fingar Peter, Setrag Khoshafian, Larry Hawes, in: Layna Fischer (Ed.), BPM Everywhere: Internet of Things, Process of Everything, first ed., Future Strategies Inc., Lighthouse Point, FL, USA, 2015.
[86] Jakob Mass, Chii Chang, Satish N. Srirama, Workflow model distribution or code distribution? ideal approach for service composition of the internet of things, in: International Conference on Services Computing, SCC, IEEE, 2016, pp. 649–656.
[87] Ronny Seiger, Christine Keller, Florian Niebling, Thomas Schlegel, Modelling complex and flexible processes for smart cyber-physical environments, J. Comput. Sci. 10 (2015) 137–148.
[88] Ronny Seiger, Steffen Huber, Peter Heisig, Uwe Aßmann, Toward a framework for self-adaptive workflows in cyber-physical systems, Softw. Syst. Model. 18 (2) (2019) 1117–1134.
[89] Shangguang Wang, Ao Zhou, Mingzhe Yang, Lei Sun, Ching-Hsien Hsu, Fangchun Yang, Service composition in cyber-physical-social systems, IEEE Trans. Emerg. Top. Comput. (2019) 1–11, Advance online publication.
[90] Alexander Jungmann, Felix Mohr, An approach towards adaptive service composition in markets of composed services, J. Internet Serv. Appl. 6 (1) (2015) 1–18.
[91] Mohamed Essaid Khanouche, Yacine Amirat, Abdelghani Chibani, Moussa Kerkar, Ali Yachir, Energy-centered and QoS-aware services selection for internet of things, IEEE Trans. Autom. Sci. Eng. 13 (3) (2016) 1256–1269.
[92] Shuiguang Deng, Zhengzhe Xiang, Jianwei Yin, Javid Taheri, Albert Y. Zomaya, Composition-driven IoT service provisioning in distributed edges, IEEE Access 6 (2018) 54258–54269.
[93] Rong Yang, Bing Li, Can Cheng, A Petri net-based approach to service composition and monitoring in the IOT, in: Asia-Pacific Services Computing Conference, APSCC, IEEE, 2014, pp. 16–22.
[94] Mengyu Sun, Zhensheng Shi, Shengjun Chen, Zhangbing Zhou, Yucong Duan, Energy-efficient composition of configurable internet of things services, IEEE Access 5 (2017) 25609–25622.
[95] Osama Alsaryrah, Ibrahim Mashal, Tein-Yaw Chung, Bi-objective optimization for energy aware internet of things service composition, IEEE Access 6 (2018) 26809–26819.
[96] Lei Huo, Zhiliang Wang, Service composition instantiation based on cross-modified artificial Bee Colony algorithm, China Commun. 13 (10) (2016) 233–244.
[97] Zhangbing Zhou, Deng Zhao, Lu Liu, Patrick C.K. Hung, Energy-aware composition for wireless sensor networks as a service, Future Gener. Comput. Syst. 80 (2018) 299–310.
[98] Qian Li, Runliang Dou, Fuzan Chen, Guofang Nan, A QoS-oriented Web service composition approach based on multi-population genetic algorithm for Internet of things, Int. J. Comput. Intell. Syst. 7 (sup2) (2014) 26–34.
[99] Safina Showkat Ara, Zia Ush Shamszaman, Ilyoung Chong, Web-of-objects based user-centric semantic service composition methodology in the internet of things, Int. J. Distrib. Sens. Netw. 10 (5) (2014) 1–11.
[100] Mahmoud M. Badawy, Zainab H. Ali, Hesham A. Ali, Qos provisioning framework for service-oriented internet of things (IoT), Cluster Comput. (2019) 1–17, Advance online publication.
[101] Samir Berrani, Ali Yachir, Badis Djemaa, Mohamed Aissani, Extended multi-agent system based service composition in the internet of things, in: International Conference on Pattern Analysis and Intelligent Systems, PAIS, IEEE, 2018, pp. 1–8.

[102] A. Urbieta, A. González-Beltrán, S. Ben Mokhtar, M. Anwar Hossain, L. Capra, Adaptive and context-aware service composition for IoT-based smart cities, Future Gener. Comput. Syst. 76 (2017) 262–274.
[103] Tomasz Szydlo, Robert Brzoza-Woch, Joanna Sendorek, Mateusz Windak, Chris Gniady, Flow-based programming for IoT leveraging fog computing, in: International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE, IEEE, 2017, pp. 74–79.
[104] Jonathan Lee, Shin-Jie Lee, Ping-Feng Wang, A framework for composing SOAP, non-SOAP and non-web services, IEEE Trans. Serv. Comput. 8 (2) (2015) 240–250.
[105] Farzad Khodadadi, Amir Vahid Dastjerdi, Rajkumar Buyya, Simurgh: A framework for effective discovery, programming, and integration of services exposed in IoT, in: International Conference on Recent Advances in Internet of Things, RIoT, IEEE, 2015, pp. 1–6.
[106] Panagiotis Vlacheas, Raffaele Giaffreda, Vera Stavroulaki, Dimitris Kelaidonis, Vassilis Foteinos, George Poulios, Panagiotis Demestichas, Andrey Somov, Abdur Rahim Biswas, Klaus Moessner, Enabling smart cities through a cognitive management framework for the internet of things, IEEE Commun. Mag. 51 (6) (2013) 102–111.
[107] Hongming Cai, Yizhi Gu, Athanasios V. Vasilakos, Boyi Xu, Jianzhong Zhou, Model-driven development patterns for mobile services in cloud of things, IEEE Trans. Cloud Comput. 6 (3) (2018) 771–784.
[108] In-Young Ko, Han-Gyu Ko, Angel Jimenez Molina, Jung-Hyun Kwon, SoIoT: Toward a user-centric IoT-based service framework, ACM Trans. Internet Technol. 16 (2) (2016) 1–21.
[109] Alfred Åkesson, Görel Hedin, Mattias Nordahl, Boris Magnusson, ComPOS: Composing oblivious services, in: International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops, IEEE, 2019, pp. 132–138.
[110] Fabrizio Montesi, Claudio Guidi, Gianluigi Zavattaro, Service-oriented programming with Jolie, in: Athman Bouguettaya, Quan Z. Sheng, Florian Daniel (Eds.), Web Services Foundations, Springer, New York, NY, USA, 2014, pp. 81–107.
[111] Le Kim-Hung, Soumya Kanti Datta, Christian Bonnet, François Hamon, Alexandre Boudonne, A scalable IoT framework to design logical data flow using virtual sensor, in: International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob, IEEE, 2017, pp. 1–7.
[112] Anne H. Ngu, Mario Gutierrez, Vangelis Metsis, Surya Nepal, Quan Z. Sheng, IoT middleware: A survey on issues and enabling technologies, IEEE Internet Things J. 4 (1) (2017) 1–20.
[113] Federico Montori, Luca Bedogni, Luciano Bononi, A collaborative internet of things architecture for smart cities and environmental monitoring, IEEE Internet Things J. 5 (2) (2018) 592–605.
[114] Antonio Marcos Alberti, Gabriel Dias Scarpioni, Vaner Magalhães, Arismar Cerqueira Sodre, Joel Rodrigues, Rodrigo da Rosa Righi, Advancing novagenesis architecture towards future internet of things, IEEE Internet Things J. 6 (1) (2019) 215–229.
[115] Sandra Rodríguez-Valenzuela, Juan A. Holgado-Terriza, José M. Gutiérrez-Guerrero, Jesús L. Muros-Cobos, Distributed service-based approach for sensor data fusion in IoT environments, Sensors 14 (10) (2014) 19200–19228.
[116] Robert Kleinfeld, Stephan Steglich, Lukasz Radziwonowicz, Charalampos Doukas, Glue.things: A mashup platform for wiring the internet of things with the internet of services, in: International Workshop on Web of Things, WoT, ACM, 2014, pp. 16–21.
[117] Sylvain Cherrier, Ismail Salhi, Yacine M. Ghamri-Doudane, Stéphane Lohier, Philippe Valembois, BeC 3: Behaviour crowd centric composition for IoT applications, Mob. Netw. Appl. 19 (1) (2014) 18–32.
[118] Alexis Huf, Frank Siqueira, Composition of heterogeneous web services: A systematic review, J. Netw. Comput. Appl. 143 (2019) 89–110.
[119] Parvaneh Asghari, Amir Masoud Rahmani, Hamid Haj Seyyed Javadi, Internet of things applications: A systematic review, Comput. Netw. 148 (2019) 241–261.
[120] Xabier Larrucea, Izaskun Santamaria, Ricardo Colomo-Palacios, Christof Ebert, Microservices, IEEE Software 35 (3) (2018) 96–100.
[121] Cloves Carneiro, Tim Schmelmer, Microservices: The what and the why, in: Cloves Carneiro, Tim Schmelmer (Eds.), Microservices from Day One, Apress, Berkeley, CA, USA, 2016, pp. 3–18.
[122] Sam Newman, Building Microservices, first ed., O'Reilly Media, Sebastopol, CA, USA, 2015.
[123] Ion-Dorinel Filip, Florin Pop, Cristina Serbanescu, Chang Choi, Microservices scheduling model over heterogeneous cloud-edge environments as support for IoT applications, IEEE Internet Things J. 5 (4) (2018) 2672–2681.
[124] Ahmed E. Khaled, Abdelsalam Helal, Wyatt Lindquist, Choonhwa Lee, IoT-DDL–device description language for the "t" in IoT, IEEE Access 6 (2018) 24048–24063.
[125] Lorenzo Carnevale, Antonio Celesti, Antonino Galletta, Schahram Dustdar, Massimo Villari, From the cloud to edge and IoT: a smart orchestration architecture for enabling osmotic computing, in: International Conference on Advanced Information Networking and Applications Workshops, WAINA, IEEE, 2018, pp. 419–424.
[126] Yuansong Qiao, Robert Nolani, Saul Gill, Guiming Fang, Brian Lee, ThingNet: A micro-service based IoT macro-programming platform over edges and cloud, in: Conference on Innovation in Clouds, Internet and Networks and Workshops, ICIN, IEEE, 2018, pp. 1–4.
[127] Kleanthis Thramboulidis, Danai C. Vachtsevanou, Alexandros Solanos, Cyber-physical microservices: An IoT-based framework for manufacturing systems, in: Industrial Cyber-Physical Systems, ICPS, IEEE, 2018, pp. 232–239.
[128] Olaf Zimmermann, Microservices tenets, Comput. Sci. - Res. Dev. 32 (3) (2017) 301–310.
[129] J. Paul Morrison, Flow-Based Programming: A New Approach to Application Development, second ed., CreateSpace, Paramount, CA, USA, 2010.
[130] Wesley M. Johnston, J. R. Paul Hanna, Richard J. Millar, Advances in dataflow programming languages, ACM Comput. Surv. 36 (1) (2004) 1–34.
[131] J. Paul Morrison, Data stream linkage mechanism, IBM Syst. J. 17 (4) (1978) 383–408.
[132] Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju, Web services, in: Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju (Eds.), Web Services: Concepts, Architectures and Applications, Springer, Berlin, Germany, 2004, pp. 123–149.
[133] Nam Ky Giang, Michael Blackstock, Rodger Lea, Victor C.M. Leung, Developing IoT applications in the fog: A distributed dataflow approach, in: International Conference on the Internet of Things, IOT, IEEE, 2015, pp. 155–162.
[134] Joseph Noor, Hsiao-Yun Tseng, Luis Garcia, Mani Srivastava, Ddflow: Visualized declarative programming for heterogeneous IoT networks, in: International Conference on Internet of Things Design and Implementation, IoTDI, ACM, 2019, pp. 172–177.
[135] National Instruments, What is labview? 2019, URL http://www.ni.com/en-gb/shop/labview.html.
[136] Chris Peltz, Web services orchestration and choreography, Computer 36 (10) (2003) 46–52.
[137] Alistair Barros, Marlon Dumas, Phillipa Oaks, Standards for web service choreography and orchestration: status and perspectives, in: Christop J. Bussler, Armin Haller (Eds.), Business Process Management Workshops, in: Lecture Notes in Computer Science, vol. 3812, Springer, Berlin, Germany, 2005, pp. 61–74.
[138] Quan Sheng, Xiaoqiang Qiao, Athanasios Vasilakos, Claudia Szabo, Scott Bourne, Xiaofei Xu, Web services composition: A decade's overview, Inform. Sci. 280 (2014) 218–238.
[139] Pieter Hens, Monique Snoeck, Geert Poels, Manu De Backer, Process fragmentation, distribution and execution using an event-based interaction scheme, J. Syst. Softw. 89 (2014) 170–192.
[140] Ward Jaradat, Alan Dearle, Adam Barker, Towards an autonomous decentralized orchestration system, Concurr. Comput.: Pract. Exper. 28 (11) (2016) 3164–3179.
[141] Mangala Gowri Nanda, Satish Chandra, Vivek Sarkar, Decentralizing execution of composite web services, in: ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA, ACM, 2004, pp. 170–187.
[142] Girish Chafle, Sunil Chandra, Vijay Mann, Mangala Gowri Nanda, Decentralized orchestration of composite web services, in: International World Wide Web Conference, WWW, ACM, 2004, pp. 134–143.
[143] Michael Pantazoglou, Ioannis Pogkas, Aphrodite Tsalgatidou, Decentralized enactment of BPEL processes, IEEE Trans. Serv. Comput. 7 (2) (2014) 184–197.
[144] Daniel Wutke, Daniel Martin, Frank Leymann, Model and infrastructure for decentralized workflow enactment, in: ACM Symposium on Applied Computing, SAC, ACM, 2008, pp. 90–94.
[145] Daniel Martin, Daniel Wutke, Frank Leymann, Tuplespace middleware for Petri net-based workflow execution, Int. J. Web Grid Serv. 6 (1) (2010) 35–57.
[146] Mirko Sonntag, Katharina Gorlach, Dimka Karastoyanova, Frank Leymann, Michael Reiter, Process space-based scientific workflow enactment, Int. J. Bus. Process Integr. Manage. 5 (1) (2010) 32–44.
[147] Xitong Kang, Xudong Liu, Hailong Sun, Yanjiu Huang, Chao Zhou, Improving performance for decentralized execution of composite web services, in: World Congress on Services, SERVICES, IEEE, 2010, pp. 582–589.
[148] Walid Fdhila, Marlon Dumas, Claude Godart, Luciano García-Bañuelos, Heuristics for composite Web service decentralization, Softw. Syst. Model. 13 (2) (2014) 599–619.

[149] Ronny Seiger, Steffen Huber, Thomas Schlegel, PROtEUS: An integrated system for process execution in cyber-physical systems, in: Khaled Gaaloul, Rainer Schmidt, Selmin Nurcan, Sérgio Guerreiro, Qin Ma (Eds.), Enterprise, Business-Process and Information Systems Modeling, in: Lecture Notes in Business Information Processing, vol. 214, Springer, Cham, Switzerland, 2015, pp. 265–280.
[150] Yongyang Cheng, Shuai Zhao, Bo Cheng, Junliang Chen, A service-based fog execution environment for the IoT-aware business process applications, in: International Conference on Web Services, ICWS, IEEE, 2018, pp. 323–326.
[151] Amazon, AWS step functions, 2019, URL http://aws.amazon.com/step-functions/.
[152] Adam Barker, Christopher D. Walton, David Robertson, Choreographing web services, IEEE Trans. Serv. Comput. 2 (2) (2009) 152–166.
[153] Gero Decker, Oliver Kopp, Alistair Barros, An introduction to service choreographies, Inf. Technol. 52 (2) (2008) 122–127.
[154] OW2 Consortium, CHOReVOLUTION, URL http://www.chorevolution.eu.
[155] S. Cherrier, R. Langar, Services organisation in IoT: mixing orchestration and choreography, in: Global Information Infrastructure and Networking Symposium, GIIS, IEEE, 2018, pp. 1–4.
[156] ActnConnect, The choreographic platform Actorsphere, 2019, URL http://actnconnect.de/actorsphere_en.
[157] Damian Arellanes, Kung-Kiu Lau, Workflow variability for autonomic IoT systems, in: International Conference on Autonomic Computing, ICAC, IEEE, 2019.
[158] Damian Arellanes, Kung-Kiu Lau, Algebraic service composition for user-centric IoT applications, in: Dimitrios Georgakopoulos, Liang-Jie Zhang (Eds.), Internet of Things – ICIOT 2018, in: Lecture Notes in Computer Science, vol. 10972, Springer, Cham, Switzerland, 2018, pp. 56–69.
[159] Damian Arellanes, Kung-Kiu Lau, Exogenous connectors for hierarchical service composition, in: International Conference on Service-Oriented Computing and Applications, SOCA, IEEE, 2017, pp. 125–132.
[160] Kung-Kiu Lau, Perla Velasco Elizondo, Zheng Wang, Exogenous connectors for software components, in: George T. Heineman, Ivica Crnkovic, Heinz W. Schmidt, Judith A. Stafford, Clemens Szyperski, Kurt Wallnau (Eds.), Component-Based Software Engineering, in: Lecture Notes in Computer Science, vol. 3489, Springer, Berlin, Germany, 2005, pp. 90–106.
[161] Perla Velasco Elizondo, Kung-Kiu Lau, A catalogue of component connectors to support development with reuse, J. Syst. Softw. 83 (7) (2010) 1165–1178.
[162] Damian Arellanes, Kung-Kiu Lau, Decentralized data flows in algebraic service compositions for the scalability of IoT systems, in: World Forum on Internet of Things, WF-IoT, IEEE, 2019, pp. 668–673.
[163] Damian Arellanes, Kung-Kiu Lau, D-XMAN: A platform for total compositionality in service-oriented architectures, in: International Symposium on Cloud and Service Computing, SC2, IEEE, 2017, pp. 283–286.
[164] T. Lin, H. Rivano, F. Le Mouël, A survey of smart parking solutions, IEEE Trans. Intell. Transp. Syst. 18 (12) (2017) 3229–3253.
[165] Tullio Giuffrè, Sabato Marco Siniscalchi, Giovanni Tesoriere, A novel architecture of parking management for smart cities, Procedia - Soc. Behav. Sci. 53 (2012) 16–28.
[166] Jan Höller, Vlasios Tsiatsis, Catherine Mulligan, Stamatis Karnouskos, Stefan Avesand, David Boyle, From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence, first ed., Academic Press, Oxford, UK, 2014.
[167] Tariq Samad, Anuradha Annaswamy, The Impact of Control Technology: Overview, Success Stories, and Research Challenges, Technical Report, IEEE Control Systems Society, 2011, p. 175.
[168] Jasmine Sekhon, Cody Fleming, Towards improved testing for deep learning, in: International Conference on Software Engineering, ICSE, IEEE, 2019.
[169] P. Barros, R. Just, S. Millstein, P. Vines, W. Dietl, M. dAmorim, M.D. Ernst, Static analysis of implicit control flow: Resolving java reflection and android intents (T), in: International Conference on Automated Software Engineering, ASE, IEEE, 2015, pp. 669–679.
[170] Animesh Nandi, Atri Mandal, Shubham Atreja, Gargi B. Dasgupta, Subhrajit Bhattacharya, Anomaly detection using program control flow graph mining from execution logs, in: International ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD, ACM, 2016, pp. 215–224.
[171] Fábio Bezerra, Jacques Wainer, W.M.P. van der Aalst, Anomaly detection using process mining, in: Terry Halpin, John Krogstie, Selmin Nurcan, Erik Proper, Rainer Schmidt, Pnina Soffer, Roland Ukor (Eds.), Enterprise, Business-Process and Information Systems Modeling, in: Lecture Notes in Business Information Processing, vol. 29, Springer, Berlin, Germany, 2009, pp. 149–161.
[172] Tigist Abera, N. Asokan, Lucas Davi, Jan-Erik Ekberg, Thomas Nyman, Andrew Paverd, Ahmad-Reza Sadeghi, Gene Tsudik, C-FLAT: Control-flow attestation for embedded systems software, in: ACM SIGSAC Conference on Computer and Communications Security, ACM, 2016, pp. 743–754.
[173] Christian Collberg, Clark Thomborson, Douglas Low, Manufacturing cheap, resilient, and stealthy opaque constructs, in: ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL, ACM, 1998, pp. 184–196.
[174] Babak Yadegari, Jon Stephens, Saumya Debray, Analysis of exception-based control transfers, in: ACM Conference on Data and Application Security and Privacy, CODASPY, ACM, 2017, pp. 205–216.
[175] Zipkin, Openzipkin . A distributed tracing system, 2019, URL https://zipkin.io/.
[176] Alexandre Bergel, Felipe Bañados, Romain Robbes, Walter Binder, Execution profiling blueprints, Softw. - Pract. Exp. 42 (9) (2012) 1165–1192.
[177] Joshua Garcia, Daniel Popescu, Gholamreza Safi, William G.J. Halfond, Nenad Medvidovic, Identifying message flow in distributed event-based systems, in: Joint Meeting on Foundations of Software Engineering, ESEC/FSE, ACM, 2013, pp. 367–377.
[178] A. Burattin, A. Sperduti, W.M.P. van der Aalst, Control-flow discovery from event streams, in: IEEE Congress on Evolutionary Computation, CEC, IEEE, 2014, pp. 2420–2427.
[179] Wil van der Aalst, Process Mining: Data Science in Action, second ed., Springer, Berlin, Germany, 2016.
[180] Volodymyr Leno, Abel Armas-Cervantes, Marlon Dumas, Marcello La Rosa, Fabrizio M. Maggi, Discovering process maps from event streams, in: International Conference on Software and System Process, ICSSP, ACM, 2018, pp. 86–95.
[181] Olin Grigsby Shivers, Control-flow Analysis of Higher-order Languages of Taming Lambda (Ph. D. thesis), Carnegie Mellon University, 1991.
[182] Ryan Cunningham, Eddie Kohler, Making events less slippery with eel, in: Conference on Hot Topics in Operating Systems, HOTOS, USENIX Association, 2005, pp. 1–6.
[183] Josh Evans, Mastering chaos - a netflix guide to microservices, 2016, URL https://www.infoq.com/presentations/netflix-chaos-microservices.
[184] Payam Barnaghi, Amit Sheth, On searching the internet of things: Requirements and challenges, IEEE Intell. Syst. 31 (6) (2016) 71–75.
[185] Zhenyu Wen, Renyu Yang, Peter Garraghan, Tao Lin, Jie Xu, Michael Rovatsos, Fog orchestration for internet of things services, IEEE Internet Comput. 21 (2) (2017) 16–24.
[186] Michele Zorzi, Alexander Gluhak, Sebastian Lange, Alessandro Bassi, From today's INTRAnet of things to a future INTERnet of things: a wireless- and mobility-related view, IEEE Wirel. Commun. 17 (6) (2010) 44–51.
[187] Jörg Heuer, Johannes Hund, Oliver Pfaff, Toward the web of things: Applying web technologies to the physical world, Computer 48 (5) (2015) 34–42.
[188] Samir Tata, Rakesh Jain, Heiko Ludwig, Sandeep Gopisetty, Living in the cloud or on the edge: Opportunities and challenges of IOT application architecture, in: International Conference on Services Computing, SCC, IEEE, 2017, pp. 220–224.
[189] Adam Barker, Rajkumar Buyya, Decentralised orchestration of service-oriented scientific workflows, in: International Conference on Cloud Computing and Services Science, CLOSER, SciTePress, 2011, pp. 222–231.
[190] Tanveer Ahmed, Abhinav Tripathi, Abhishek Srivastava, Rain4service: An approach towards decentralized web service composition, in: International Conference on Services Computing, SCC, IEEE, Anchorage, AK, USA, 2014, pp. 267–274.
[191] T. Ahmed, M. Mrissa, A. Srivastava, Magel: A magneto-electric effect-inspired approach for web service composition, in: International Conference on Web Services, ICWS, IEEE, 2014, pp. 455–462.
[192] Walid Fdhila, Ustun Yildiz, Claude Godart, A flexible approach for automatic process decentralization using dependency tables, in: International Conference on Web Services, ICWS, IEEE, 2009, pp. 847–855.
[193] M. Schmidt, B. Hutchison, P. Lambros, R. Phippen, The Enterprise Service Bus: Making service-oriented architecture real, IBM Syst. J. 44 (4) (2005) 781–797.
[194] Nicolai Josuttis, Soa in Practice: The Art of Distributed System Design, first ed., O'Reilly Media, Sebastopol, CA, USA, 2007.
[195] Walter Binder, Ion Constantinescu, Boi Faltings, Service invocation triggers: a lightweight routing infrastructure for decentralised workflow orchestration, Int. J. High Perform. Comput. Netw. 6 (1) (2009) 81–90.
[196] Janggwan Im, Seonghoon Kim, Daeyoung Kim, IoT mashup as a service: Cloud-based mashup service for the internet of things, in: International Conference on Services Computing, SCC, IEEE, 2013, pp. 462–469.
[197] Felipe Pontes Guimaraes, Eduardo Hideo Kuroda, Daniel Macedo Batista, Performance evaluation of choreographies and orchestrations with a new simulator for service compositions, in: International Workshop on Computer Aided Modeling and Design of Communication Links and Networks, CAMAD, IEEE, 2012, pp. 140–144.
[198] David Liu, Data-flow distribution in FICAS service composition infrastructure, in: International Conference on Parallel and Distributed Computing Systems, PDCAT, 2002, pp. 1–6.

[199] Jan Seeger, Rohit A. Deshmukh, Vasil Sarafov, Arne Bröring, Dynamic IoT choreographies, IEEE Pervasive Comput. 18 (1) (2019) 19–27.
[200] Mahda Noura, Mohammed Atiquzzaman, Martin Gaedke, Interoperability in internet of things: Taxonomies and open challenges, Mob. Netw. Appl. 24 (3) (2019) 796–809.
[201] Frank Leymann, Web services: Distributed applications without limits - an outline, in: Database Systems for Business, Technology and Web, BTW, 2003, pp. 1–22.
[202] Kung-Kiu Lau, Vladyslav Ukis, Perla Velasco, Zheng Wang, A component model for separation of control flow from computation in component-based systems, in: International Workshop on Aspect-Based and Model-Based Separation of Concerns in Software Systems, ABMB, 2006, pp. 57–69.
[203] Farhad Arbab, Composition of interacting computations, in: Dina Goldin, Scott A. Smolka, Peter Wegner (Eds.), Interactive Computation, Springer, Berlin, Germany, 2006, pp. 277–321.
[204] Ahmed Safwat, M.B. Senousy, Addressing challenges of ultra large scale system on requirements engineering, in: International Conference on Communications, Management, and Information Technology, ICCMIT'2015, 2015, pp. 442–449.
[205] Mehdi Mirakhorli, Amir Azim Sharifloo, Fereidoon Shams, Architectural challenges of ultra large scale systems, in: International Workshop on Ultra-Large-Scale Software-Intensive Systems, ULSSIS, ACM, 2008, pp. 45–48.
[206] Andrea Ceccarelli, Andrea Bondavalli, Bernhard Froemel, Oliver Hoeftberger, Hermann Kopetz, Basic concepts on systems of systems, in: Andrea Bondavalli, Sara Bouchenak, Hermann Kopetz (Eds.), Cyber-Physical Systems of Systems: Foundations – a Conceptual Model and Some Derivations: the AMADEOS Legacy, Springer, Cham, Switzerland, 2016, pp. 1–39.
[207] Amel Bennaceur, Ciaran McCormick, Jesús García-Galán, Charith Perera, Andrew Smith, Andrea Zisman, Bashar Nuseibeh, Feed me, feed me: An exemplar for engineering adaptive software, in: International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS, IEEE, 2016, pp. 89–95.
[208] Marcello La Rosa, Wil M.P. Van Der Aalst, Marlon Dumas, Fredrik P. Milani, Business process variability modeling: A survey, ACM Comput. Surv. 50 (1) (2017) 1–45.
[209] Official government website for Northern Ireland citizens, Off-street and on-street parking, 2019, URL https://www.nidirect.gov.uk/articles/street-and-street-parking.
[210] Wei Fan, Zhengyong Chen, Zhang Xiong, Hui Chen, The Internet of data: a new idea to extend the IOT in the digital world, Front. Comput. Sci. 6 (6) (2012) 660–667.
[211] Richard N. Taylor, Nenad Medvidovic, Eric M. Dashofy, Software Architecture: Foundations, Theory, and Practice, first ed., Wiley Publishing, Hoboken, NJ, USA, 2009.
[212] Paolo Nesi, Gianni Pantaleo, Michela Paolucci, Imad Zaza, Auditing and assessment of data traffic flows in an IoT architecture, in: International Conference on Collaboration and Internet Computing, CIC, IEEE, 2018, pp. 388–391.
[213] Giacomo Ghidini, Sajal K. Das, Vipul Gupta, Fuseviz: A framework for web-based data fusion and visualization in smart environments, in: International Conference on Mobile Ad-Hoc and Sensor Systems, MASS, IEEE, 2012, pp. 468–472.
[214] Juan Luis Pérez, Álvaro Villalba, David Carrera, Iker Larizgoitia, Vlad Trifa, The COMPOSE API for the internet of things, in: International Conference on World Wide Web, WWW, ACM, 2014, pp. 971–976.
[215] Antonio Pintus, Davide Carboni, Andrea Piras, Paraimpu: a platform for a social web of things, in: International Conference on World Wide Web, WWW, ACM, 2012, pp. 401–404.
[216] Spacebrew, What is spacebrew? 2019, URL spacebrew.cc.
[217] Gennaro De Luca, Zhongtao Li, Sami Mian, Yinong Chen, Visual programming language environment for different IoT and robotics platforms in computer science education, CAAI Trans. Intell. Technol. 3 (2) (2018) 119–130.
[218] Per Persson, Ola Angelsmark, Calvin – merging cloud and IoT, in: International Conference on Ambient Systems, Networks and Technologies, ANT, Elsevier, 2015, pp. 210–217.
[219] Intel, IoT services orchestration layer, 2019, URL https://github.com/intel/intel-iot-services-orchestration-layer.
[220] NoFlo, NoFlo: Flow-based programming for javascript, 2019, URL noflojs.org.
[221] Gábor Paller, Endri Bezati, Nebojša Taušan, Gábor Farkas, Gábor Élő, Dataflow-based Heterogeneous Code Generator for IoT Applications, IEEE, 2019, pp. 428–434.
[222] Yuuichi Teranishi, Takashi Kimata, Hiroaki Yamanaka, Eiji Kawai, Hiroaki Harai, Dynamic data flow processing in edge computing environments, in: Annual Computer Software and Applications Conference, COMPSAC, IEEE, 2017, pp. 935–944.
[223] Marco Mesiti, Stefano Valtolina, Luca Ferrari, Minh-Son Dao, Koji Zettsu, An editable live ETL system for ambient intelligence environments, in: World Forum on Internet of Things, WF-IoT, IEEE, 2015, pp. 393–394.
[224] Mahmoud Hussein, Shuai Li, Ansgar Radermacher, Model-driven development of adaptive IoT systems, in: International Workshop on Interplay of Model-Driven and Component-Based Software Engineering, ModComp, 2017, pp. 20–27.
[225] Christian Prehofer, Ilias Gerostathopoulos, Modeling restful web of things services: Concepts and tools, in: Quan Z. Sheng, Yongrui Qin, Lina Yao, Boualem Benatallah (Eds.), Managing the Web of Things, Morgan Kaufmann, Boston, MA, USA, 2017, pp. 73–104.
[226] Andrei Ciortea, Olivier Boissier, Antoine Zimmermann, Adina Magda Florea, Responsive decentralized composition of service mashups for the internet of things, in: International Conference on the Internet of Things, IoT, in: IoT'16, ACM, 2016, pp. 53–61.
[227] Florian Daniel, Maristella Matera, Mashups: Concepts, Models and Architectures, first ed., in: Data-Centric Systems and Applications, Springer, Heidelberg, Germany, 2014.
[228] Michael Blackstock, Rodger Lea, IoT mashups with the WoTKit, in: International Conference on the Internet of Things, IoT, IEEE, 2012, pp. 1–8.
[229] Florian Daniel, Barbara Pernici, Insights into web service orchestration and choreography:, Int. J. E-Bus. Res. 2 (1) (2006) 58–77.
[230] Patrik Spiess, Stamatis Karnouskos, Dominique Guinard, Domnic Savio, Oliver Baecker, Luciana Moreira Sá De Souza, Vlad Trifa, SOA-based integration of the internet of things in enterprise services, in: International Conference on Web Services, ICWS, IEEE, 2009, pp. 968–975.
[231] Antonio Pintus, Davide Carboni, Andrea Piras, Alessandro Giordano, Connecting smart things through web services orchestrations, in: Florian Daniel, Federico Michele Facca (Eds.), Current Trends in Web Engineering, in: Lecture Notes in Computer Science, vol. 6385, Springer, Berlin, Germany, 2010, pp. 431–441.
[232] Nils Glombitza, Sebastian Ebers, Dennis Pfisterer, Stefan Fischer, Using BPEL to realize business processes for an internet of things, in: Hannes Frey, Xu Li, Stefan Ruehrup (Eds.), Ad-Hoc, Mobile, and Wireless Networks, in: Lecture Notes in Computer Science, vol. 6811, Springer, Berlin, Germany, 2011, pp. 294–307.
[233] Hagen Overdick, Towards resource-oriented BPEL, in: Thomas Gschwind, Cesare Pautasso (Eds.), Emerging Web Services Technology, Birkhäuser, Basel, Switzerland, 2008, pp. 129–140.
[234] Feng Chen, Changrui Ren, Qinhua Wang, Bing Shao, A process definition language for internet of things, in: International Conference on Service Operations and Logistics, and Informatics, SOLI, IEEE, 2012, pp. 107–110.
[235] Dulce Domingos, Francisco Martins, Ricardo Martinho, Mário Silva, Ad-hoc changes in IoT-aware business processes, in: Internet of Things, IOT, IEEE, 2010, pp. 1–7.
[236] OMG, Business process model and notation (BPMN) version 2.0, 2011, URL https://www.omg.org/spec/BPMN/2.0/.
[237] Camunda, BPMN workflow engine, 2019, URL https://camunda.com/products/bpmn-engine/.
[238] Tijs Rademakers, Activiti in Action: Executable Business Processes in BPMN 2.0, first ed., Manning Publications, Greenwich, CT, USA, 2012.
[239] Minjae Park, Hyunah Kim, Hyun Ahn, Kwanghoon Pio Kim, A process-aware IoT application execution environment design, in: International Conference on Advanced Communication Technology, ICACT, IEEE, 2018, pp. 724–727.
[240] Gero Decker, Oliver Kopp, Frank Leymann, Mathias Weske, BPEL4chor: Extending BPEL for modeling choreographies, in: International Conference on Web Services, ICWS, IEEE, 2007, pp. 296–303.
[241] Steve Ross-Talbot, Tony Fletcher, Web services choreography description language: Primer, 2006, URL https://www.w3.org/TR/ws-cdl-10-primer/.
[242] Johannes Maria Zaha, Alistair Barros, Marlon Dumas, Arthur ter Hofstede, Let's dance: A language for service behavior modeling, in: On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, in: Lecture Notes in Computer Science, vol. 4275, Springer, Berlin, Germany, 2006, pp. 145–162.
[243] Assaf Arkin, Sid Askary, Scott Fordin, Web service choreography interface (WSCI) 1.0, 2002, URL https://www.w3.org/TR/2002/NOTE-wsci-20020808/.
[244] Gero Decker, Alistair Barros, Interaction modeling using BPMN, in: Arthur ter Hofstede, Boualem Benatallah, Hye-Young Paik (Eds.), Business Process Management Workshops, in: Lecture Notes in Computer Science, vol. 4928, Springer, Berlin, Germany, 2007, pp. 208–219.
[245] Michael Zimmermann, Uwe Breitenbücher, Frank Leymann, A TOSCA-based programming model for interacting components of automatically deployed cloud and IoT applications, in: International Conference on Enterprise Information Systems, ICEIS, SciTePress, 2017, pp. 121–131.

[246] Andreas Weiß, Vasilios Andrikopoulos, Santiago Gómez Sáez, Michael Hahn, Dimka Karastoyanova, Chorsystem: A message-based system for the life cycle management of choreographies, in: Christophe Debruyne, Hervé Panetto, Robert Meersman, Tharam Dillon, eva Kühn, Declan O'Sullivan, Claudio Agostino Ardagna (Eds.), On the Move to Meaningful Internet Systems: OTM 2016 Conferences, in: Lecture Notes in Computer Science, vol. 10033, Springer, Cham, Switzerland, 2016, pp. 503–521.
[247] F.L. Traversa, M. Di Ventra, Universal memcomputing machines, IEEE Trans. Neural Netw. Learn. Syst. 26 (11) (2015) 2702–2715.
[248] A. Rajhans, A. Bhave, I. Ruchkin, B.H. Krogh, D. Garlan, A. Platzer, B. Schmerl, Supporting heterogeneity in cyber-physical systems architectures, IEEE Trans. Automat. Control 59 (12) (2014) 3178–3193.
[249] Tomas Bures, Ilias Gerostathopoulos, Petr Hnetynka, Jaroslav Keznikl, Michal Kit, Frantisek Plasil, DEECO: An ensemble-based component system, in: International ACM Sigsoft Symposium on Component-Based Software Engineering, CBSE, ACM, 2013, pp. 81–90.
[250] Ikuo Nakagawa, Masahiro Hiji, Hiroshi Esaki, Dripcast - architecture and implementation of server-less java programming framework for billions of IoT devices, J. Inform. Process. 23 (4) (2015) 458–464.
[251] Y. Zhang, J.L. Chen, B. Cheng, Integrating events into SOA for IoT services, IEEE Commun. Mag. 55 (9) (2017) 180–186.
[252] F. Halili, E. Rufati, I. Ninka, Styles of service composition – analysis and comparison methods, in: International Conference on Computational Intelligence, Communication Systems and Networks, CICSyN, IEEE, 2013, pp. 302–307.

Damian Arellanes received his Ph.D. degree in Computer Science from the University of Manchester, UK, and holds an M.Sc. degree in Computer Science from the Center for Research and Advanced Studies of the National Polytechnic Institute (CINVESTAV-IPN). He is currently a Research Associate at the Department of Computer Science, The University of Manchester. Damian won the best paper award at the 2018 International Conference on Internet of Things (ICIOT), and his research interests include service composition, Internet of Things and autonomic computing.

Kung-Kiu Lau holds a Ph.D. from the University of Leeds, UK, and is currently a senior lecturer, leading the Component-based Software Development research group, in the School of Computer Science, the University of Manchester, UK. He is the author of a textbook, the editor of a research monograph and a book series on Component-based Software Development, all published by World Scientific. He is a member of the editorial boards of the Journal of Universal Computer Science and the Journal of Applied Logic (area editor: Logic and Software Engineering).