Decentralised Data Flows for the Functional Scalability of Service-Oriented IoT Systems

Damian Arellanes[1], Kung-Kiu Lau[2] and Rizos Sakellariou[2]

[1]*School of Computing and Communications, Lancaster University, Lancaster LA1 4WA, United Kingdom*
[2]*Department of Computer Science, The University of Manchester, Manchester M13 9PL, United Kingdom*
Email: damian.arellanes@lancaster.ac.uk

Horizontal and vertical scalability have been widely studied in the context of computational resources. However, with the exponential growth in the number of connected objects, functional scalability (in terms of the size of software systems) is rapidly becoming a central challenge for building efficient service-oriented IoT systems that generate huge volumes of data continuously. As systems scale up, a centralised approach for moving data between services becomes infeasible because it leads to a single performance bottleneck. A distributed approach avoids such a bottleneck but it incurs additional network traffic as data streams pass through multiple mediators. Decentralised data exchange is the only solution for realising totally efficient IoT systems, since it avoids a single performance bottleneck and dramatically minimises network traffic. In this paper, we present a functionally scalable approach that separates data and control for the realisation of decentralised data flows in service-oriented IoT systems. Our approach is evaluated empirically, and the results show that it scales well with the size of IoT systems by substantially reducing both the number of data flows and network traffic in comparison with distributed data flows.

Keywords: Internet of Things; Decentralised Data Flows; DX-MAN; Algebraic Service Composition; Functional Scalability; Separation of Control and Data

1. INTRODUCTION

The Internet of Things (IoT) is penetrating essential domains of our daily life such as healthcare, security or city management. This emerging paradigm promises the seamless interconnection of any physical object (i.e., thing) through innovative distributed services, leading to service-oriented IoT systems.[3] A proper service composition mechanism is thus required for the integration of IoT services into workflows that consist of both control flow and data flow [1]. Control flow refers to the order in which services are executed, whilst data flow defines how services move data over the network.

As of early 2021, there are over 20 billion connected things and it is predicted that this number will exponentially grow in the next few years [2, 3, 4]. Hence, as IoT systems may potentially consist of an overwhelmingly large number of services, *functional scalability* becomes a challenging concern. Unlike vertical and horizontal scalability, functional scalability *accommodates growth in terms of the number of services composed in an IoT system* [1]. To tackle the functional scalability challenge, a service composition mechanism must provide the means to compose loosely-coupled services that exchange data as efficiently as possible over the network.

IoT services can exchange data in three different ways: (i) with a centralised coordinator, (ii) with multiple distributed coordinators or (iii) in a purely decentralised manner. A *centralised approach* [5] relies on a single coordinator to mediate data streams between services. Although this is viable for enterprise scenarios where small amounts of data are involved, the coordinator would easily become a bottleneck in IoT systems that generate data in the order of Petabytes. To avoid the central bottleneck, a *distributed approach* [6] can be used for balancing loads over multiple coordinators. However, this would cause network performance degradation as data passes through multiple mediators before reaching the actual data consumers.

A *decentralised approach* is the most efficient way to exchange data as it enables direct data exchanges

[3]For the rest of the paper, the terms *IoT system* and *service-oriented IoT system* are used interchangeably.

from producers to consumers, thereby decreasing latency and maximising throughput [7, 8, 9, 10, 11, 12, 13, 14]. However, exchanging data among loosely coupled IoT services is not trivial, especially in resource-constrained environments where things have poor network connection and low disk space [7]. Furthermore, building data dependency graphs is hard when control flow and data flow are tightly coupled (i.e., when data follows control). To overcome such issues, recent research [15, 16, 17, 8, 18, 1] shows that the separation of control and data can be useful since it allows independent reasoning, monitoring, maintenance and evolution of control flow and data flow. Consequently, an efficient data exchange approach can be defined without considering control, so a reduced number of messages can be transmitted over the network.

1.1. Related Work

This section presents and analyses related work on decentralised data flows in service-oriented systems. We classify the existing approaches into three major categories, depending on the composition mechanism they are built on: (i) orchestration (with coordinated data exchanges), (ii) P2P dataflows and (iii) P2P choreography.

In orchestration-based approaches [19, 20, 21, 11], a central orchestrator coordinates system execution by passing data references alongside control. Although data values do not follow control like in traditional orchestration [22, 23], additional network traffic is introduced since references and acknowledgement messages are routed via the network. This additional network traffic arises from the fact that orchestration-based approaches do not separate data and control.

Dataflows is a composition mechanism that builds a graph of data transformations, where vertices receive data, perform some computation and pass the result(s) to other vertices via data flow edges [24, 25]. Such data exchanges can be done with or without mediators. In P2P dataflows [14, 26, 27, 28, 29], services exchange data with no mediators between them [1]. As control flow is implicit in the collaborative exchange of data, P2P dataflows do not separate data and control.

Service choreography describes decentralised interactions among participants using a well-defined public protocol. In P2P choreography, atomic services are participants that exchange data via direct message passing [30]. Some approaches [13, 31] introduce proxies to coordinate the invocation of services and the exchange of data alongside control. In any case, P2P choreography does not separate data and control.

Table 1 summarises our analysis of related work, where it is clear that there is no approach supporting the separation of control and data for realising decentralised data flows between atomic services, in order to efficiently support environments that generate huge volumes of data continuously. Furthermore, existing approaches only consider vertical and horizontal scalability. It is important to mention that the separation of concerns does not imply decentralisation but provides a number of benefits. Particularly, the separation is necessary to avoid passing references alongside control during system execution, thus, reducing the number of messages transmitted over the network. Also, the orthogonality of data and control enables separate reasoning, monitoring, maintenance, reuse and evolution of those concerns, as discussed in other studies [1, 8].

In the analysis above, we did not consider decentralised orchestration, coordinated dataflows and choreographed orchestration, since those approaches do not support decentralised data flows between atomic services. Nevertheless, for completeness, we analyse those categories below.

In approaches built on top of decentralised orchestration, multiple composite services coordinate system execution [10, 9, 32, 12]. In particular, approaches like [32] persist data on distributed data spaces which may become a bottleneck in IoT environments where services exchange huge volumes of data continuously. Although some works (e.g., [12]) solve the bottleneck issue by persisting references instead of values, they require the maintenance of tuple spaces and databases for moving references and storing data, respectively. Moving references is necessary because decentralised orchestration does not separate data and control. Paradoxically, decentralised orchestration does not support decentralised dataflows between atomic services.

In coordinated dataflows [33], a master composite coordinates data flows between slave composites. As data passes through multiple mediators, this approach introduces unnecessary network hops (like decentralised orchestration). This problem is also present in [34] which introduces the notion of abstract data types for moving data outside services. Discovery Net [35] and Kepler [36] are workflow systems that implement the semantics of coordinated dataflows, which provide mechanisms to coordinate the execution of (disjoint) data flow graphs. Discovery Net allows the definition of a control flow graph where nodes exchange control tokens instead of data values. Nodes can be data flow graphs *per se*. In Kepler, a director is a semantic entity that controls data exchanges from outside a data flow graph. These systems do not support decentralised data flows because their workflow execution mechanisms are centralised [9].

In choreographed orchestration, data passes through multiple orchestrator participants leading to network degradation [37]. To separate control and data, [8] proposes a choreographed orchestration approach that allows the manual modelling of cross-partner data dependencies. However, there are no decentralised data flows between atomic services, but only between partners.

TABLE 1: Analysis of related work.

Approach	Separation of Data and Control	Functional Scalability	Decentralised Dataflows
Orchestration with Coordinated Data Exchanges	×	×	✓
P2P Dataflows	×	×	✓
P2P Choreography	×	×	✓
Our Approach	✓	✓	✓

1.2. Paper Contributions

This paper proposes a functionally scalable approach that semantically separates data and control, in order to decentralise data flows between atomic services in IoT systems. Accordingly, this paper makes the following contributions:

- We propose an (implementation-independent) service composition model that semantically separates data and control by so-called *data forests*. The semantics allows the encapsulation of explicit control flow graphs and explicit data dependency graphs. As graphs are orthogonal, the semantics enables a separate reasoning and analysis of data and control.
- We propose an efficient algorithm that leverages the model semantics for a compositional analysis of *data forests*. The algorithm analyses a data dependency graph, without considering control flow, for the automatic construction of a decentralised mapping between data consumers and data producers. The algorithm reduces both the number of data flows and network traffic of an IoT system.
- Unlike the state-of-the-art on decentralised data flows, we evaluate the proposed approach in terms of functional scalability. The evaluation is implementation-independent and shows that the approach scales well with the number of services by reducing the number of data flows linearly and network traffic logarithmically.

1.3. Paper Organisation

The rest of the paper is organised as follows. Section 2 presents a motivating scenario to explain why decentralised data flows are a crucial desideratum in the IoT domain. Section 3 presents an overview of the model semantics and a detailed description of the proposed approach. Section 4 presents the implementation of the proposed solution. Section 5 presents a case study. Section 6 presents a comparative evaluation of decentralised data flows versus distributed data flows in the case study. Section 7 outlines an evaluation of the proposed approach in terms of functional scalability. Section 8 outlines the conclusions. Appendices A and B formalise our motivating scenario. Appendix C describes an example of the deployment-time process. Appendix D presents a source code fragment of our implementation. Appendix E presents the notation used throughout the paper.

2. WHY DO WE NEED DECENTRALISED DATA FLOWS IN IOT?

This section discusses the rationale for enabling decentralised data flows in IoT, with the help of a scenario based on the smart parking system presented in [1] and real-world occupancy data from car parks in Birmingham, UK (see Appendix A). The system is used by drivers who want to find the nearest parking space while they drive around a smart city. The workflow for this scenario is a combination of the *reduce* and the *sequential* data patterns and involves the services shown in Fig. 1. Although data passes through operations, we consider that it is routed between services as there is a one-to-one relationship between services and operations. For simplicity, we assume that services are deployed on different things, e.g., the *Availability Checking Service* is deployed on an infostation while the *Space Finding Service* is deployed on an edge router. An infostation is an urban infrastructure device that collects up-to-date status from all occupancy sensors in range. We assume that all parking spaces are equipped with occupancy sensors whose functionality is abstracted by a *Sensor Service*.

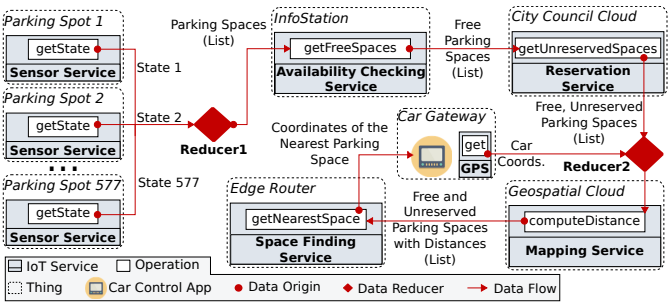


FIGURE 1: Workflow of the Smart Parking System.

The workflow of our scenario is triggered by a driver's request, and it starts with two parallel tasks. In the first one, the *Data Reducer 2* receives the driver's coordinates from the *GPS Service*. In the second one, the *Data Reducer 1* pulls the states from multiple *Sensor* services to create a list that shows the status of parking spaces. Then, the *Availability Checking Service* uses the list to determine which parking spaces are free. As some spaces can be reserved in advance, the *Reservation Service* determines which free parking spaces are unreserved, and passes its resulting list to the *Data Reducer 2*. Once the *Data Reducer 2* receives all its inputs, it forwards them to the *Mapping Service*. The *Mapping Service*

then computes the distance between each (free and unreserved) space and the driver's location, and appends the distances to the list. Finally, the *Space Finding Service* determines the nearest parking space for the driver, using the calculations from the *Mapping Service*. The coordinates of the nearest parking space are the final result of the workflow, and are passed to the car control application.

As the number of occupancy sensors could be huge, especially in large cities, the services of the smart parking system may potentially exchange vast amounts of data continuously. This is because there is a one-to-one mapping between sensors and services, and occupancy sensors are frequently producing data. Thus, functional scalability becomes a challenging concern. To understand how we can address this problem, we analyse the three existing approaches for passing data between the services of our scenario (see Fig. 2). For simplicity, we do not show control flow and we assume that there is no data required for both pulling sensor data and starting the execution of the system (from the car control application).

In our scenario, there are 373 drivers arriving to the car park *BHMBCCMKT01* between 07:59:45 and 16:26:47 on a given day (see Appendix A). This car park has 577 occupancy sensors each producing 100 bytes, so there are $577 \cdot 100 = 57.7$ KB of sensor data per driver request. The number of bytes returned by the *Availability Checking Service*, the *Reservation Service* and the *Mapping Service* are computed by the functions $S_A(t)$, $S_R(t)$ and $S_M(t)$, whose definitions are presented in Appendix B. For simplicity, we assume that the *GPS Service* and the *Space Finding Service* always return 40 bytes for the coordinates of a driver and a nearest space, respectively.

We refer the reader to Appendix A for details about the data set and the formalisms used in our scenario. Appendix B presents the calculations for the total data transmitted over the network, considering that there are 373 drivers between 07:59:45 and 16:26:47. These calculations are presented for each data exchange approach depicted in Fig. 2. Below we interpret the results.

A centralised approach [38, 39, 5] depends on a single coordinator for passing data between services. Fig. 2(a) shows how a central coordinator mediates data streams for the smart parking system. The main drawback of this approach is that such a coordinator is a potential bottleneck (as huge amounts of data are exchanged) and introduces an extra network hop for passing data. This approach is therefore a threat for scalability as pointed out by many researchers [7, 8, 11, 14]. Furthermore, IoT may potentially require the deployment of coordinators on resource-constrained things (e.g., edge devices) which can lead to bottlenecks due to the presence of low computing power and poor network resources [7]. For instance, the central coordinator of our scenario is deployed on an edge device which manages ~102.98MB

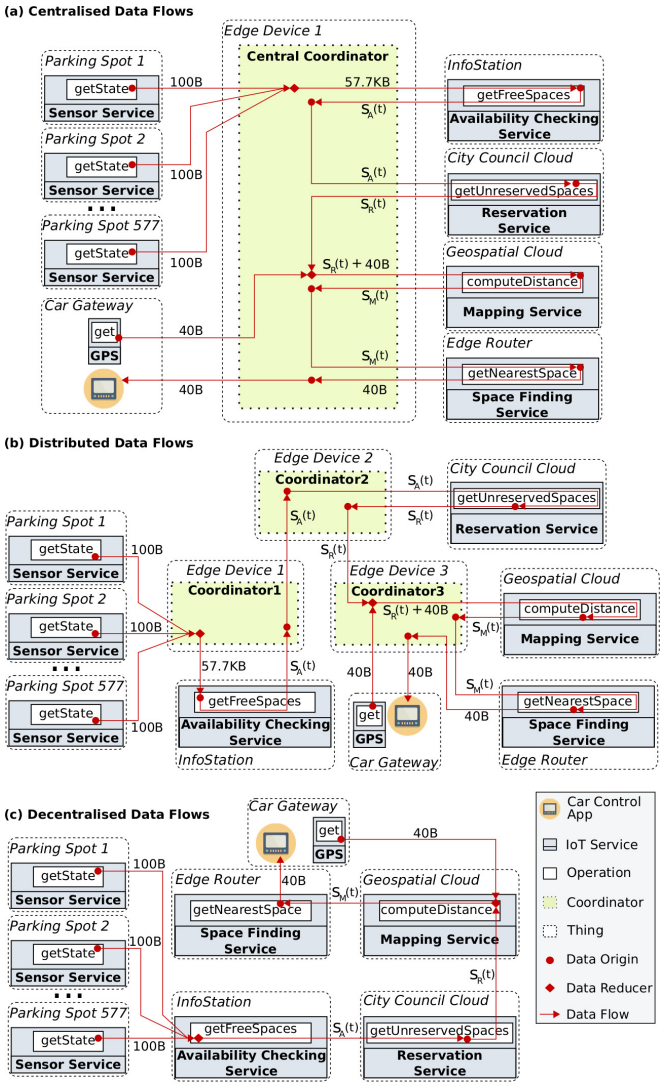


FIGURE 2: Data flow approaches for the Smart Parking System.

of data over the network using a poor connection (see Equation 5 in Appendix B). Certainly, this impacts the overall performance of the system negatively. This is especially true in large cities with multiple drivers requesting a space at the same time.

Although a distributed approach [40, 41, 33] removes the central coordinator, it introduces unnecessary network traffic as data passes through multiple mediators, even if data is unimportant for them. For example, in Fig. 2(b) the data generated by the *Availability Checking Service* goes through *Coordinator1* and then through *Coordinator2*, before reaching the service that really needs the data (i.e., the *Reservation Service*). These additional network hops negatively impact the overall performance of the system.

For our scenario, the distribution of data load among three coordinators avoids a single bottleneck. However, as coordinators mediate interactions, ~123.36 MB of data is exchanged over the network (see Equation 5

in Appendix B), leading to ~19.80% more network traffic than the centralised approach (see at the bottom of Appendix B). In fact, this indicates that the network traffic may increase linearly with the number of coordinators.

The decentralised data exchange approach [7, 8, 11, 13] is the most efficient one, since data is passed directly to the services that actually need it. Particularly, it requires only one network hop to pass data from a data producer to a data consumer, as depicted in Fig. 2(c). The decentralised version of our scenario requires a total data transfer of ~51.49MB over the network (see Equation 7 in Appendix B), i.e., ~50.00% less network traffic than the centralised approach and ~58.26% less traffic than the distributed one (see at the bottom of Appendix B).

3. THE DX-MAN MODEL

To realise decentralised data flows, we propose to extend the semantics of DX-MAN [42, 43, 44], which is an algebraic model for building IoT systems, where services and exogenous composition operators are first-class semantic entities (see Fig. 3).

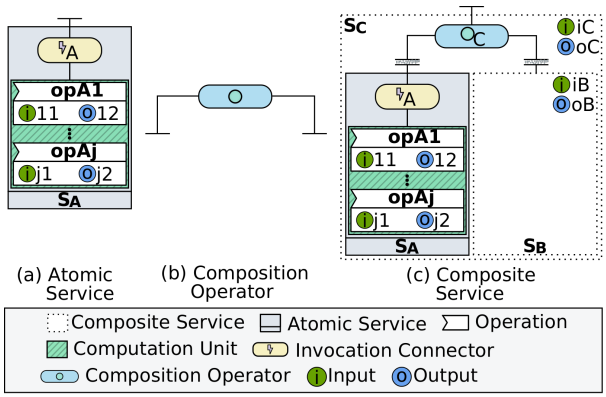


FIGURE 3: DX-MAN Model.

A *service* $S \in \mathbb{S}$ is a stateless distributed unit of composition that can be atomic or composite. Its semantics is a *workflow space* $W \in \mathbb{W}$ which is a (finite or infinite) set of workflow control flow variants $w_{i\in[1,\infty]} \in W$ that represent alternative service behaviours:

$$\mathbb{S} \equiv \mathbb{W} \equiv \{w_1, w_2, \ldots\} \tag{1}$$

where $\mathbb{S}$ is the service type and $\mathbb{W}$ is the workflow space type.

An atomic service is syntactically formed by connecting an invocation connector with a computation unit which encapsulates the implementation of multiple operations and it is not allowed to invoke other units (see Fig. 3(a)). Semantically, this is equivalent to constructing an *atomic workflow space* whose variants invoke different operations in the computation unit via the invocation connector. An operation performs some computation and has an input parameter and an output parameter.

The DX-MAN model relies on algebraic composition for defining complex services. Algebraic composition is the process by which a *composition operator* hierarchically composes multiple services of type $\mathbb{S}$ into a *composite service* of type $\mathbb{S}$ which, in turn, can be further composed into even more complex composites (see Fig. 3(b)) [44]. A composition operator defines control flows exogenously (i.e., outside the composed services) to avoid service dependencies. Formally, it is a function $\circ$ for n services with the following type:

$$\circ : \mathbb{S}^n \rightarrow \mathbb{S} \tag{2}$$

A composite service is syntactically formed by connecting a composition operator with multiple (atomic and/or composite) services, which has an input parameter and an output parameter (see Fig. 3(c)). Semantically, it is a *composite workflow space* produced by a composition operator taking sub-workflow spaces as operands. There are composition operators for sequencing (i.e., sequencer), branching (i.e., inclusive selector and exclusive selector) and parallelism (i.e., paralleliser). A sequencer or a paralleliser defines an infinite workflow space, whilst a branching operator defines a finite one.

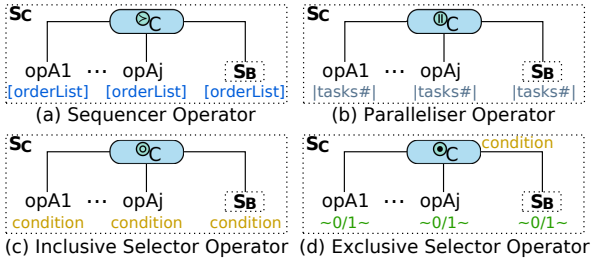


FIGURE 4: Abstract workflow trees for the composite service shown in Fig. 3, when $\circ_C$ is (a) a sequencer operator, (b) a paralleliser operator, (c) an inclusive selector operator or (d) an exclusive selector operator.

The control flow structure of a composite service is represented by an abstract workflow tree whose leaves are operations in atomic sub-services, whole composite sub-services or any combination thereof (see Fig. 4). An abstract workflow tree allows the definition of a particular variant from a composite workflow space. To do so, a concrete workflow tree must be created, which is a selection function over a set of workflow variants, and it is therefore isomorphic to an abstract workflow tree. The edges of a concrete workflow tree are labelled according to the composition operator used. In particular, the label of a sequencer edge is an ordered list of natural numbers and the label of a paralleliser edge is a natural number (representing the amount of parallel tasks). For the edges of an inclusive selector and an exclusive selector, the labels are conditions and

boolean values, respectively. An exclusive selector also has an associated global condition. For further details on workflow selection, see [42, 43].

3.1. Separation of Control Flows and Data Flows

IoT service composition can be endogenous or exogenous, depending on control flow origin [45]. Endogenous composition is used by P2P choreographies which mix service computation (i.e., operations) with control flow constructs (see Fig. 5(a)). By contrast, exogenous composition relies on coordinators that define control flow outside the computation of composed services (see Fig. 5(b)). This composition approach is used by orchestration and by DX-MAN. Unlike its counterpart, exogenous composition avoids control flow dependencies between services and facilitates reuse at scale [46, 47, 29, 44, 36]. It also avoids application logic being embedded in the computation of multiple atomic services, thereby facilitating tracking and monitoring which are crucial desiderata of functional scalability [1, 48].

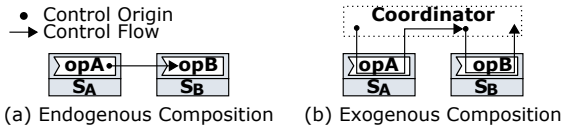


FIGURE 5: Endogenous composition vs exogenous composition.

Despite the above advantages, exogenous composition suffers from performance issues when data follows control [11, 14, 49]. This is because coordinators mediate data streams even for data that is unimportant for them (see Figs. 2(a) and 2(b)).[4] To address this problem, we propose to define control flow and data flow as orthogonal dimensions (see Fig. 6). The idea is that coordinators (i.e., composition operators) never exchange data during workflow execution, but only control.

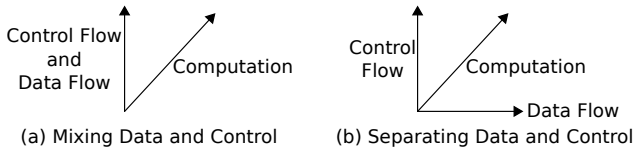


FIGURE 6: Possible DX-MAN dimensions. This paper proposes an approach for realising (b).

In DX-MAN, a workflow is defined by a concrete workflow tree (see Fig. 7(b)). It has an input parameter and an output parameter, and describes a series of steps for the invocation of operations in atomic services, sub-workflows or any combination thereof.[5] Fig. 7(c) shows an example of a sequential workflow for the invocation of the operation *opA* (provided by the atomic service S_A) and the operation *opB* (provided by the atomic service S_B). This workflow variant results from the composition depicted in Fig. 7(a) where the atomic services S_B and S_C are composed into the composite S_D by the sequencer operator $\circ_D$. Likewise, the sequencer operator $\circ_E$ composes S_A and S_D into the top-level composite S_E. For each composite, we select a workflow with a different concrete workflow tree. Note that a DX-MAN composition is done in a hierarchical bottom-up manner and, although there are infinite sequential workflow variants, we describe only one of them for the sake of this paper. Also note that the workflow invoking *opB* is a sub-workflow of the top-level one. This is because S_D is composed into S_E.

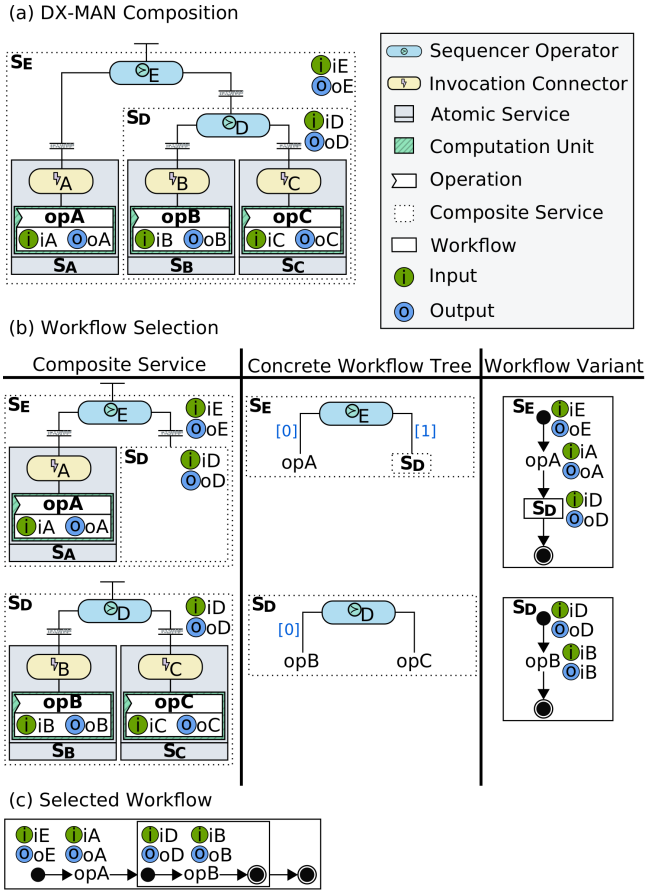


FIGURE 7: Relationship between DX-MAN compositions and IoT workflows.

A workflow execution traverses a control flow tree from top to bottom and then returns backwards (see Fig. 8). Thus, the execution of the workflow depicted in Fig. 7(c) starts with the activation of $\circ_E$ which sequentially passes control to the invocation connector

[4] Section 2 provides a running example to describe the problem of coordinated data exchanges when using exogenous composition.

[5] As a DX-MAN workflow is semantically equivalent to an orchestration, a DX-MAN composite is equivalent to a (potentially) infinite number of orchestrations.

A and then to $\circ_D$. The latter just forwards control to the invocation connector B. Once they are triggered, invocation connectors execute an operation in their computation unit.

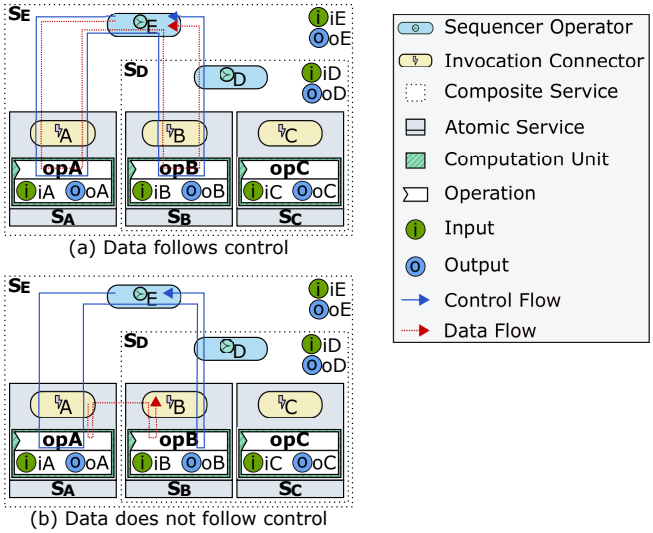


FIGURE 8: Possible executions of the workflow shown in Fig. 7(c) when control flows and data flows are (a) mixed or (b) orthogonal. In (a), composition operators (i.e., coordinators) mediate data streams, even if data is unimportant for them. In (b), data is passed directly between atomic services. This paper proposes an approach to realise (b).

If data follows control, it passes through the composition connectors $\circ_E$ and $\circ_D$ (see Fig. 8(a)). Thus, to avoid such inefficient executions, we propose an approach that enables decentralised data exchanges between atomic services by the separation of control and data (see Fig. 8(b)). Our approach is described below.

3.2. Design-time: (Semantic) Data Forest Definition

A DX-MAN system is a workflow control flow variant with an orthogonal *data forest*. The latter is a graph of directed data dependency trees defined on service composition at design-time, whose edges define an explicit relationship between parameters. Formally, a data forest is a graph $F := (V(F), E(F))$ s.t. $V(F)$ is a (finite or infinite) set of parameter vertices and $E(F)$ is a (finite or infinite) set of edges. As a vertex $v \in V(F)$ is of type $\mathbb{D}$, an edge $e \in E(F)$ is a tuple of type $\mathbb{D} \times \mathbb{D}$ where $\mathbb{D}$ is the parameter type.

For a paralleliser, the workflow input is connected to the inputs of all the invoked elements (i.e., operations or sub-workflows) whose outputs are, in turn, connected to the workflow output. In this paper, we only describe how parallel and sequential data forests are formed, since decentralised data flows are only meaningful for parallelism and sequencing. For a sequencer, the connection of vertices forms a data pipeline, so the output of an invoked element is connected to the input of the next invoked element. Additionally, the workflow input is connected to the input of the first invoked element, and the output of the last invoked element is connected to the workflow output.

Fig. 9 analyses thoroughly the composition depicted in Fig. 7. It shows the sequential workflow variants and the data forests for the composites S_D and S_E. Such forests conform to the rules described for a sequencer. The analysis is segmented because every composite in the DX-MAN model is a black box that can have any possible behaviour out of the alternative ones. Thus, as there are no (bridge) connections between data dependency trees of different forests, data is encapsulated in composite services. Therefore, composites (with all their workflows and data forests) are reusable across different IoT systems.

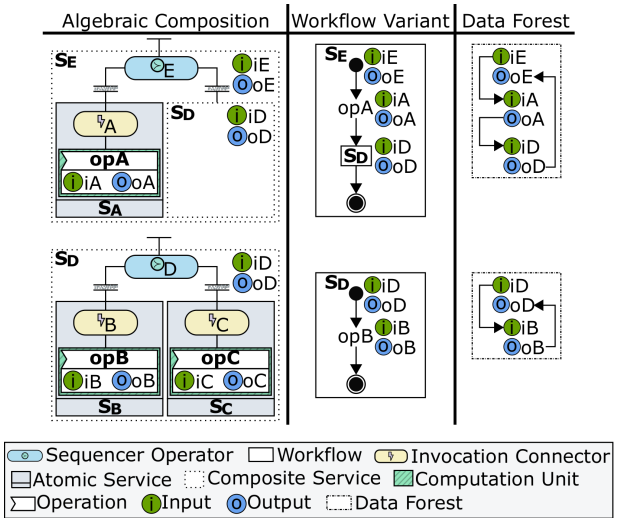


FIGURE 9: Separation of Control and Data in sequential composites. Control constructs and data dependencies are explicitly defined by exogenous composition operators and data forests, respectively.

Reusability is also present in the composite depicted in Fig. 10(a), which shows an example where the paralleliser $\circ_Z$ composes the atomic services S_X and S_Y into the composite S_Z. In this example, we define a concrete workflow tree for executing all the sub-service operations in parallel. The resulting workflow has a *fork* construct to pass control in parallel to opX and opY, and a *join* construct to synchronise control flow. Choosing this workflow implies that there is a data forest conforming to the rules previously described for a paralleliser (see Fig. 10(b)).

Although decentralised data flows are only meaningful for parallelism and sequencing, for completeness we describe how branchial data forests are formed. For both exclusive and inclusive selectors, the corresponding connection scheme is isomorphic to the one defined for a

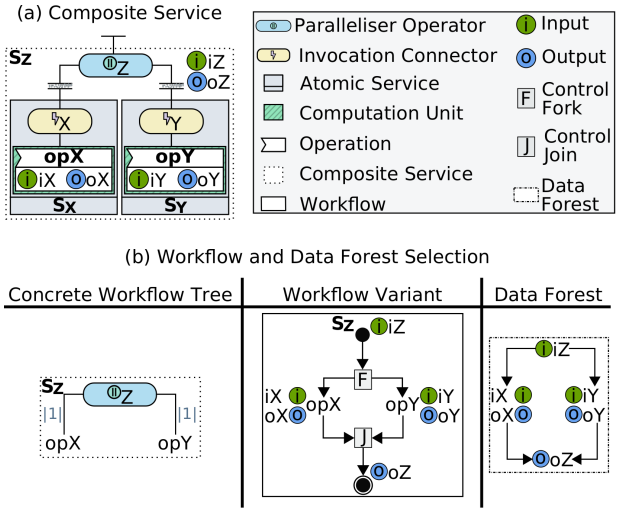


FIGURE 10: Separation of control and data in a parallel composite. Parallel constructs (i.e., fork and join) and data dependencies are explicitly defined by exogenous composition operators and data forests, respectively.

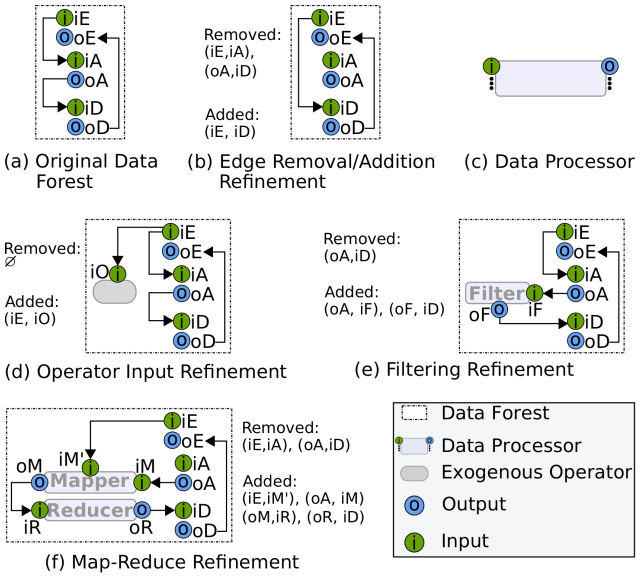


FIGURE 11: Data forest refinement.

paralleliser in which the workflow input is connected to the inputs of all the invoked elements, and the outputs of the elements are connected to the workflow output (see Fig. 10(b)). This connection is done to ensure that there is an output value no matter the execution branch taken.

3.3. Deployment-time: Data Forest Refinement

At deployment-time, a data forest can be (manually) refined in three different ways: (i) by adding or removing edges, (ii) by adding input data into exogenous operators or (iii) by introducing data processors. These refinements are entirely optional with the overall aim of providing extra flexibility for system modellers. To understand them, we consider the data forest of the composite S_E (shown in Fig. 9):

$$F_E := (\{iE, oE, iA, oA, iD, oD\}, \\ \{(iE, iA), (oA, iD), (oD, oE)\})$$

where $iE, oE, iA, oA, iD, oD \in \mathbb{D}$.

For all the scenarios, we show how edges and vertices change with respect to the original data forest (see Fig. 11(a)).

3.3.1. Edges Refinement

Refining edges is useful for changing data relationships and optimising data flows. For instance, the edge (iE, iA) can be removed when the operation opA does not need any data to perform computation. Similarly, by replacing (oA, iD) with (iE, iD), the sub-workflow S_D receives data from the top-level input instead of getting it from opA (see Fig. 11(b)). Formally, $E(F_E)$ becomes $E(F_E) \setminus \{(iE, iA), (oA, iD)\} \cup \{(iE, iD)\}$.

3.3.2. Exogenous Operator Refinement

Although they do not perform any computation, exogenous operators might require input values to evaluate boolean conditions at run-time. By default, exogenous operators do not have any parameters, but parameters can be added for a specific workflow. To do so, new vertices are added into the respective data forest and connected manually at the discretion of the modeller. Edge connections must conform to the rules described in [49]. Fig. 11(d) shows an example of this kind of refinement, where $iO \in \mathbb{D}$ represents the input parameter of the composition operator that coordinates the execution of the workflow S_E. Formally, $V(F_E)$ becomes $V(F_E) \cup \{iO\}$ and $E(F_E)$ becomes $E(F_E) \cup \{(iE, iO)\}$.

3.3.3. Data Processor Refinement

Data processors can be introduced in a data forest to perform intermediate processing. A data processor is a function that receives at least one input data, performs some custom computation and stores result(s) in at least one output (see Fig. 11(c)). By introducing data processors, we avoid tangling with the original workflow control flow of an IoT system. Although the DX-MAN model currently supports the most common data patterns (i.e., map-reduce and filtering), other data processors can be defined using the same semantics. For instance, a replicator could be introduced to copy data to multiple parameters.

Fig. 11(e) customises F_E by filtering the data generated by opA before sending it to the sub-workflow S_D. Formally, $V(F_E)$ becomes $V(F_E) \cup \{iF, oF\}$ and $E(F_E)$ becomes $E(F_E) \setminus \{(oA, iD)\} \cup \{(oA, iF), (oF, iD)\}$ s.t. $iF, oF \in \mathbb{D}$ are the filter parameters. Fig. 11(f) shows another refinement, where a mapper collects data from both opA and the parent workflow to

create a list of transformed key-value pairs. The list is then processed by a reducer which produces an output in a suitable format for S_D. Formally, $V(F_E)$ becomes $V(F_E) \cup \{iM, iM', oM, iR, oR\}$ and $E(F_E)$ becomes $E(F_E) \setminus \{(iE, iA), (oA, iD)\} \cup \{(iE, iM'), (oA, iM), (oM, iR), (oR, iD)\}$ s.t. $iM, iM', oM \in \mathbb{D}$ are the mapper parameters and $iR, oR \in \mathbb{D}$ are the reducer parameters.

3.4. Deployment-time: Data Forest Analysis

Once data forests have been defined and refined (if needed), a direct mapping between consumer parameters and producer parameters is created. To describe our approach, some formal notations are provided.

Notation 1. Let $\Pi_1 : \mathbb{D} \times \mathbb{D} \to \mathbb{D}$ be the tail map of a forest edge.

Notation 2. Let $\Pi_2 : \mathbb{D} \times \mathbb{D} \to \mathbb{D}$ be the head map of a forest edge.

Notation 3. Let $\mathbb{PI}$ be the processor input type, $\mathbb{PO}$ the processor output type, $\mathbb{OI}$ the operation input type, $\mathbb{OO}$ the operation output type, $\mathbb{WI}$ the (top-level) workflow input type, $\mathbb{WO}$ the (top-level) workflow output type and $\mathbb{EI}$ the type of an exogenous operator input s.t. $\mathbb{PI}, \mathbb{PO}, \mathbb{OI}, \mathbb{OO}, \mathbb{WI}, \mathbb{WO}, \mathbb{EI} \subset \mathbb{D}$.

Notation 4. Let V_c be the type of vertices that consume data on workflow execution, namely service operation inputs, exogenous operator inputs, data processor inputs and top-level workflow outputs s.t. $V_c \coloneqq \mathbb{OI} \cup \mathbb{EI} \cup \mathbb{PI} \cup \mathbb{WO}$. Top-level workflow outputs contain the resulting data from an IoT system execution.

Notation 5. Let V_p be the type of vertices that produce data on workflow execution (i.e., operation outputs and data processor outputs) as well as vertices representing input data of top-level workflows s.t. $V_p \coloneqq \mathbb{OO} \cup \mathbb{PO} \cup \mathbb{WI}$. Top-level workflow inputs are the data needed for an IoT system execution.

Notation 6. Let $\varsigma \subset (V_c \times V_p)$ be a binary relation (i.e., a mapping) between some consumer parameters and some producer parameters.

Notation 7. Let $\rho \subset (V_p \times V_c)$ be a binary relation (i.e., a mapping) between some producer parameters and some consumer parameters.

Our approach analyses the edges of all the data forests of an IoT system, from bottom to top with a complexity of $O(k)$ s.t. k is the total number of edges in a multi-level composition (see Fig. 12). As they do not require any data manipulation, workflow parameters (i.e., composite service parameters) are discarded by transforming the binary relations ς and ρ each time an edge is analysed. When all edges have been processed, ς is the result of our approach, i.e., a direct mapping from data consumers to data producers.

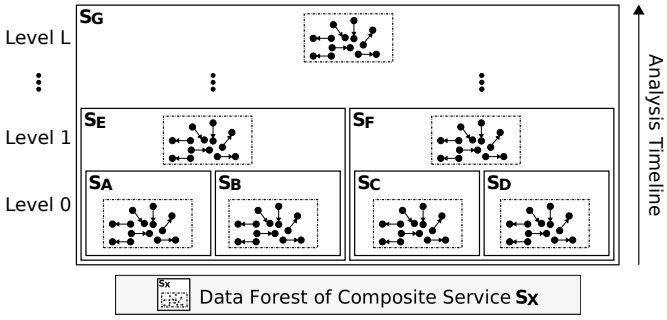


FIGURE 12: Bottom-up analysis of data forests: from level 0 to level L. The algorithm analyses the data forests of the composite services at level 0, then the data forests of the composite services at level 1, and so on until reaching the data forests of the top-level composite at level L.

We have designed Algorithm 1 to analyse individual data forest edges. The algorithm uses the sets $X_p \subset \mathbb{D}$ and $Y_c \subset \mathbb{D}$ to process the endpoints of an edge $e \in \mathbb{D} \times \mathbb{D}$. The elements of these sets are determined according to two fundamental rules. The first one is shown in lines 4-7. It states that X_p is the set of vertices connected to the *tail parameter* $\Pi_1(e)$ if the tail is not a data processor parameter and has incoming connections; otherwise, X_p only contains the tail. The second rule is shown in lines 8-13. It states that if the *head parameter* $\Pi_2(e)$ is not in a data processor and has outgoing connections, then (i) Y_c has the vertices connected from the head parameter and (ii) X_p (without the head parameter) is the set of additional producers for each consumer $y \in Y_c$. Otherwise, Y_c just contains the head parameter $\Pi_2(e)$ and the elements of X_p are additional producers of $\Pi_2(e)$. After processing the above rules, the relation ρ is updated to map each producer $x \in X_p$ to each consumer $y \in Y_c$. Appendix C presents a simple example using this algorithm. A more detailed example is presented in Section 5.

Algorithm 1 Algorithm for edge analysis

```
1:  procedure ANALYSE(e ∈ 𝔻 × 𝔻, ς, ρ)
2:      X_p ≔ ∅                                   ▷ X_p ⊂ 𝔻
3:      Y_c ≔ ∅                                   ▷ Y_c ⊂ 𝔻
4:      if Π1(e) ∉ (ℙ𝕀 ∪ ℙ𝕆) ∧ Π1(e) ∈ dom(ς) then
5:          X_p ≔ {b : (a, b) ∈ ς | a = Π1(e)}
6:      else
7:          X_p ≔ {Π1(e)}
8:      if Π2(e) ∉ (ℙ𝕀 ∪ ℙ𝕆) ∧ Π2(e) ∈ dom(ρ) then
9:          Y_c ≔ {b : (a, b) ∈ ρ | a = Π2(e)}
10:         ς ≔ ς \ {(a, Π2(e)) : a ∈ Y_c} ∪ (Y_c × X_p)
11:     else
12:         Y_c ≔ {Π2(e)}
13:         ς ≔ ς ∪ {(Π2(e), b) : b ∈ X_p}
14:     ρ ≔ ρ ∪ (X_p × Y_c)
```

4. IMPLEMENTATION

The DX-MAN platform [50] implements the semantics of the DX-MAN model and provides Java APIs to algebraically compose, deploy and execute IoT systems. Our approach is implemented on top of that platform and uses the Blockchain as the underlying decentralised data space. Please note that a Blockchain platform is just a possible implementation of a decentralised data space. Other possible implementations include OpenLink Data Spaces (ODS), ZeroMQ messaging, a shared memory for IoT or even Apache Storm.[6] We chose a Blockchain implementation to ensure data ownership for every service and data provenance for discovering data flow histories. Furthermore, we guarantee an extra layer of performance, security and auditability. It is important to note that we do not claim any contributions in terms of implementation. Our implementation is just for validation purposes.

Our approach uses *Hyperledger Fabric 1.2* as the underlying Blockchain infrastructure and *Hyperledger Composer 0.20.0* to define a Blockchain model based on three smart contracts (see Fig. 13). *CreateParameters* initialises the consumer parameters of an IoT system, while *UpdateParameters* and *ReadParameters* change and retrieve a list of parameter values from the data space, respectively. The implementation logic of the contracts is written in JavaScript and it is shown in Appendix D.

The DX-MAN platform provides the programming abstractions for composing IoT services (at design-time) and selecting IoT workflows (at deployment-time). When a workflow is selected, a data forest is automatically created for the respective control flow. Workflow selection is out of the scope of this paper; on this matter, we refer the reader to [42]. Also, to guarantee openness and replicability, the source code of the DX-MAN platform is available at `https://github.com/damianarellanes/dxman`.

Once workflows have been selected, Algorithm 1 analyses the edges of all the data forests involved to build a Java Hashmap (i.e., a binary relation ς) with a consumer parameter UUID as the key and a list of producer parameter UUIDs as the value. The entries of the map are stored as *Parameter* assets in the Blockchain using the transaction *CreateParameters*. In particular, the field *parameterId* is the hashmap key and the field *producers* is the hashmap value (see lines 7 and 12 in Fig. 13).

At run-time, exogenous operators coordinate an IoT system execution by passing control only via CoAP messages.[7] When an exogenous operator receives control, it performs one of the processes shown in Fig. 14(b-c) only if input data is needed; otherwise, steps 1-2 are omitted. When control triggers an

```
1  namespace com.dxman.blockchain
2
3  ...
4
5  asset Parameter identified by id {
6    o String id
7    o String parameterId
8    o String workflowId
9    o String value
10   o DateTime timestamp optional
11   o String updater
12   --> Parameter[] producers
13 }
14
15 concept UpdateParameterConcept {
16   --> Parameter parameter
17   o String newValue
18   o String updater
19 }
20
21 event UpdateParameterEvent {
22   o String parameter
23   o String newValue
24   o String updater
25 }
26
27 transaction CreateParameters {
28   o Parameter[] parameters
29 }
30
31 transaction UpdateParameters {
32   o UpdateParameterConcept[] updates
33 }
34
35 @returns(String[])
36 transaction ReadParameters {
37   --> Parameter[] parameters
38   o DateTime workflowTimestamp
39 }
```

FIGURE 13: Blockchain model.

invocation connector, the latter performs the process shown in Fig. 14(a). Firstly, the connector executes the transaction *ReadParameters* to retrieve the necessary inputs from the data space. For each input, the Blockchain directly consumes values from the producer list.

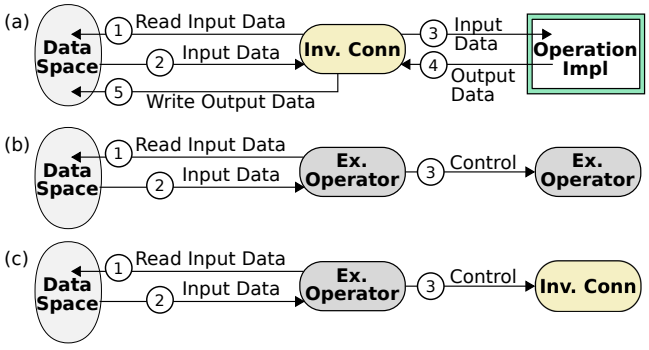


FIGURE 14: Steps performed by (a) an invocation connector and (b-c) an exogenous operator after receiving control. (b-c) are only applicable for exogenous operators that require input data. Other operators ignore the steps 1 and 2.

Producer data is stored in the Blockchain as soon as it is available, even before control reaches consumer

[6] A pipeline of different technologies can be used to efficiently implement data exchanges between services [51].

[7] `https://tools.ietf.org/html/rfc7252`

services. To avoid run-time synchronisation problems, control flow blocks in an invocation connector until all needed input values are retrieved. Next, the connector executes the computation of an operation by passing it the respective data. The result is stored in the Blockchain via the transaction *UpdateParameters*.

UpdateParameters raises an *UpdateParameterEvent* after changing a parameter value. This is useful for data processor instances that subscribe to events notified by producer parameters. Thus, a data processor instance performs a user-defined computation after receiving all the events for its inputs. Currently, our implementation supports mappers, reducers and filters, but further processors (e.g., a replicator) can be implemented conforming to the semantics presented in Section 3.3.

Consumer parameters are unaware of data producers, and vice versa, since data references are stored in the Blockchain. Furthermore, the mapping generated by Algorithm 1 avoids the inefficient approach of passing data values through exogenous operators at run-time. Therefore, as consumer parameters read data directly from producers and they are unaware of each other, our approach enables a (transparent) decentralised data exchange.

5. CASE STUDY

This section describes how to enable decentralised data flows in the smart parking scenario presented in Section 2, by covering the three phases of a DX-MAN system life-cycle.

5.1. Design-time

At design-time, we use the syntactic constructs presented in Section 3 to define a multi-level composition structure for our smart parking application (see Fig. 15). This process is done in a hierarchical bottom-up manner, starting with the composition of *Sensor Services* into *SensorNet*, via the paralleliser operator $\circ_N$. Afterwards, we use the sequencer operator $\circ_C$ to compose the *SensorNet* composite, the *Availability Checking Service* and the *Reservation Service* into *CityManagement*. In the next level, the *GPS Service* and *CityManagement* are composed into *Information* via the paralleliser operator $\circ_I$. Finally, we use the sequencer $\circ_T$ to compose *Information*, *Mapping Service* and *Space Finding Service* into *CarControlApp* (i.e., the top-level composite). Note that, according to the semantics presented in Section 3, each composite service is a workflow space with a (finite or infinite) set of workflow variants and a (finite or infinite) set of data forest variants. For clarity and conciseness, we do not present them. Instead, we refer the reader to our previous work on this matter [43, 42].

5.2. Deployment-time

For each composite, we define a concrete workflow tree to explicitly choose a workflow and a data forest from the respective space (see Section 3). In particular, for *SensorNet* there is a workflow that executes all sensor operations in parallel. For *CityManagement*, there is a workflow that sequentially invokes the *SensorNet* sub-workflow, the *getFree* operation and the *getUnreser* operation, in that order. For *Information*, there is a workflow that synchronises the execution of the *getLocation* operation and the *CityManagement* sub-workflow. Finally, in the top-level composite, the *Information* sub-workflow, the *computeDis* operation and the *getNearest* operation are triggered sequentially. The concrete workflow trees for each composite are shown in Fig. 16 and the whole workflow of our scenario is depicted in Fig. 17(a). The latter is just a concatenation of individual workflows.

In Section 3, we mentioned that data forests are implicitly selected when workflows are chosen. On that basis, the resulting data forests for each composite service are shown in Fig. 16 and the complete concatenation of data forests is illustrated in Fig. 17(b). Note that both workflows and data forests are not manually created, but just selected using the syntactic representation of concrete workflow trees.

Once data dependencies are in place, some data forests are refined (see Fig. 18(a)). In particular, we remove the edges between the inputs of the parallel composites (i.e., *SensorNet* and *Information*) and the inputs of their respective sub-services. Similarly, we remove the edges connecting the inputs of the sequential composites (i.e., *CarControlApp* and *CityManagement*) and the inputs of the parallel composites. Note that this does not mean that edges are always refined. In some scenarios, refinement never takes place.

After refinement, Algorithm 1 analyses the edges of each data forest from the bottom-level composite (i.e., *SensorNet*) to the top-level one (i.e., *CarControlApp*). The resulting binary relation between consumer parameters and producer parameters is illustrated in Fig. 18(b).

5.3. Run-time

At run-time, composition operators coordinate the execution of the selected workflow by passing control only (see Fig. 19(a)), while atomic services exchange data directly (see Fig. 19(b)). In particular, when a driver requests a nearest parking space, $\circ_T$ passes control to the $\circ_I$ operator which, in turn, forks control towards the invocation connector G and the $\circ_C$ operator. When G is triggered, it activates the *getLocation* operation before passing control back to $\circ_I$. Meanwhile, $\circ_C$ forwards control to $\circ_N$ for activating the invocation connectors of all the sensors. After receiving control from all of them, $\circ_N$ notifies the $\circ_C$ operator for executing the operations *getFree* and *getUnreser* via their invocation connectors. Once control returns from R, $\circ_C$ sends control back to $\circ_I$. If the latter has received control from G and $\circ_C$, it notifies $\circ_T$ for invoking the operations *computeDis* and

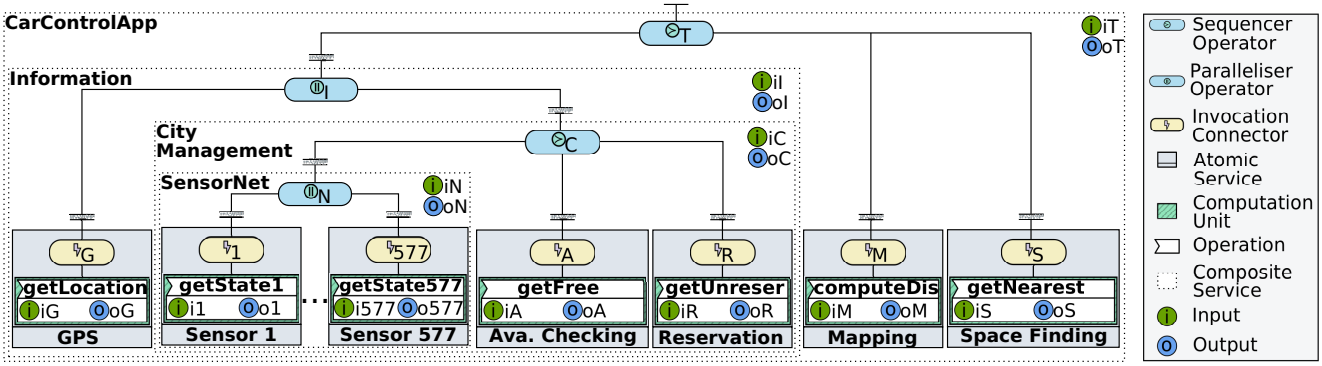


FIGURE 15: DX-MAN composition for the smart parking scenario.

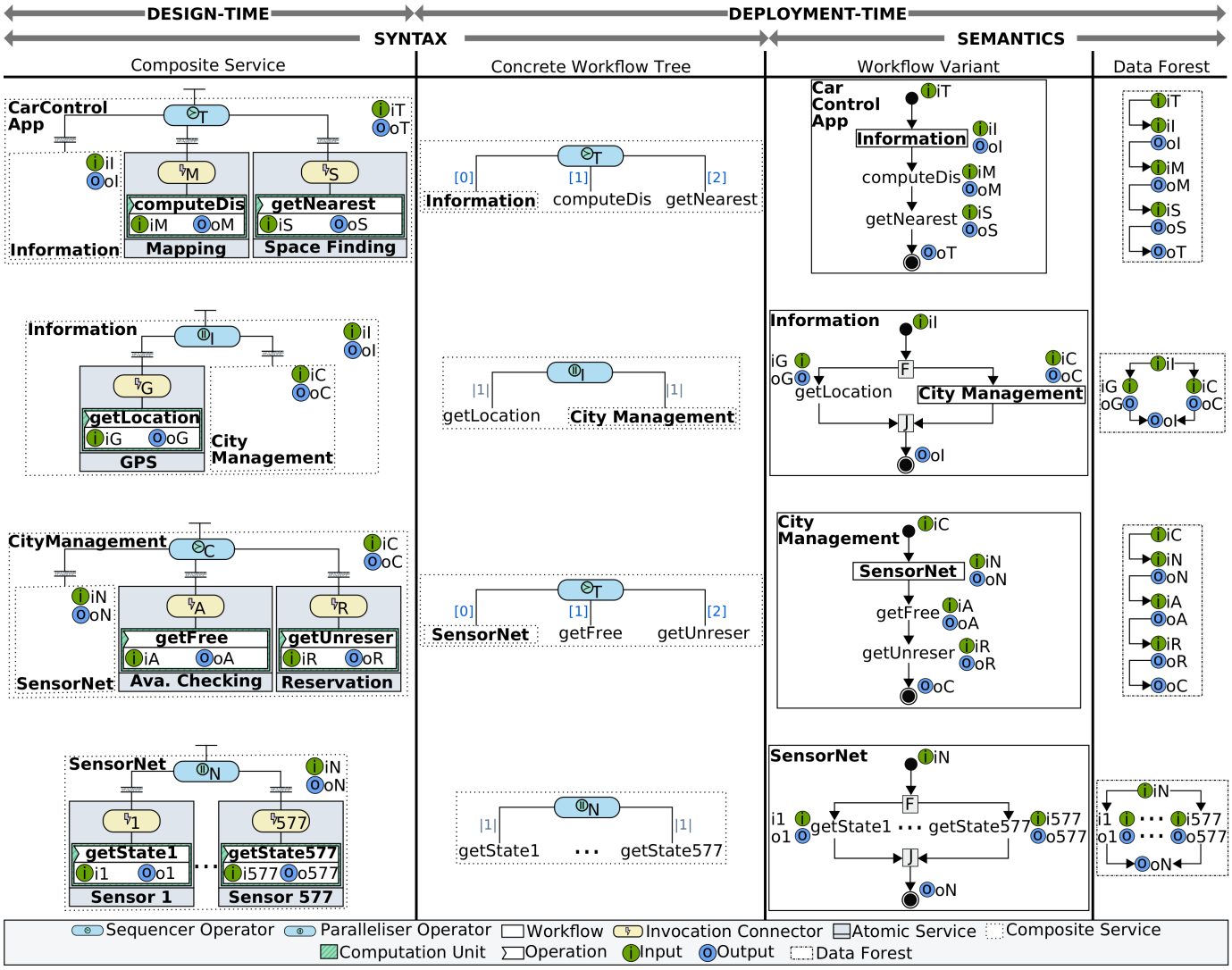


FIGURE 16: Selecting workflows explicitly and data forests implicitly for the smart parking scenario.

getNearest via M and S, respectively. The execution of the smart parking app terminates when o_T receives control from S.

Fig. 19 illustrates the execution of the smart parking scenario, where it is clear that data never passes alongside control, but it flows in a decentralised manner according to the binary relation shown in Fig. 18(b). It is important to note that control blocks in an invocation connector until all producer values become available (see Section 4). For instance, A waits until all sensor data has been produced, before executing the *getFree* operation; thereby, guaranteeing synchronisation between the *Availability Checking Service* and all *Sensor Services*. Situations like this can be common since

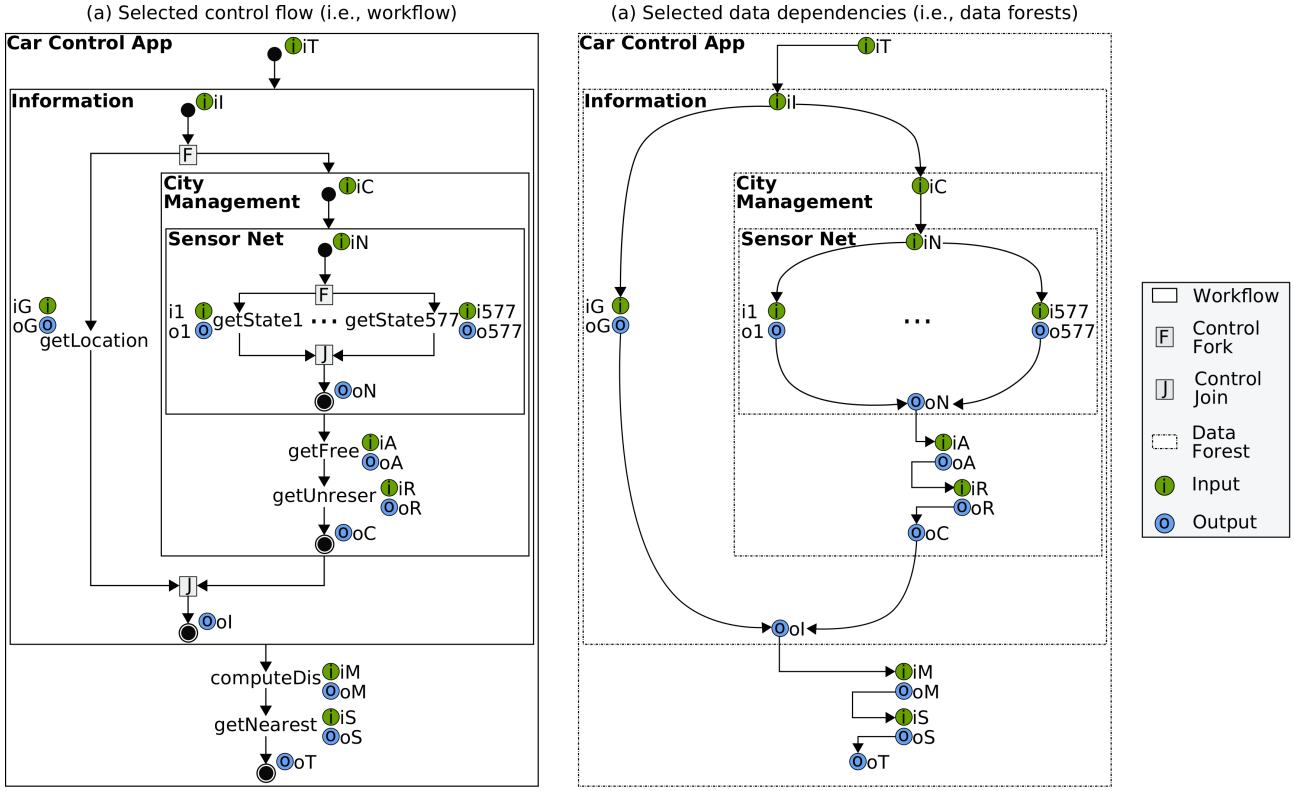


FIGURE 17: Orthogonality between control flow (i.e., workflow) and data dependencies (i.e., data forests) in the smart parking scenario.

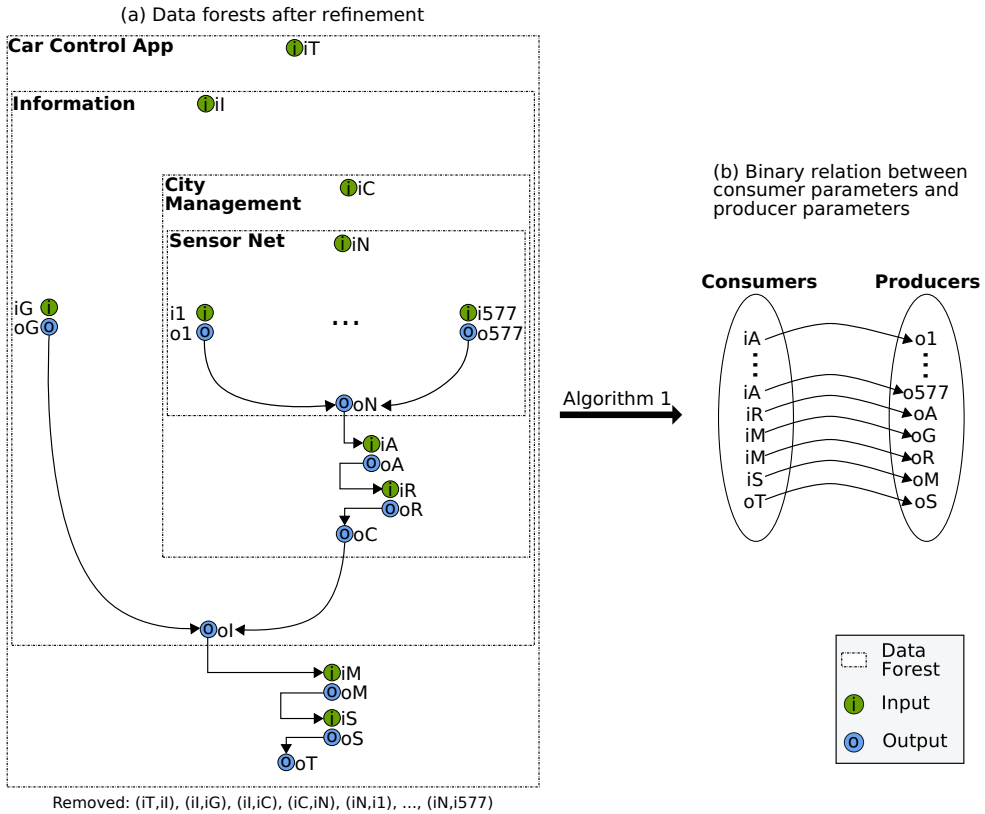


FIGURE 18: Transforming data forests into a binary relation between consumer parameters and producer parameters.

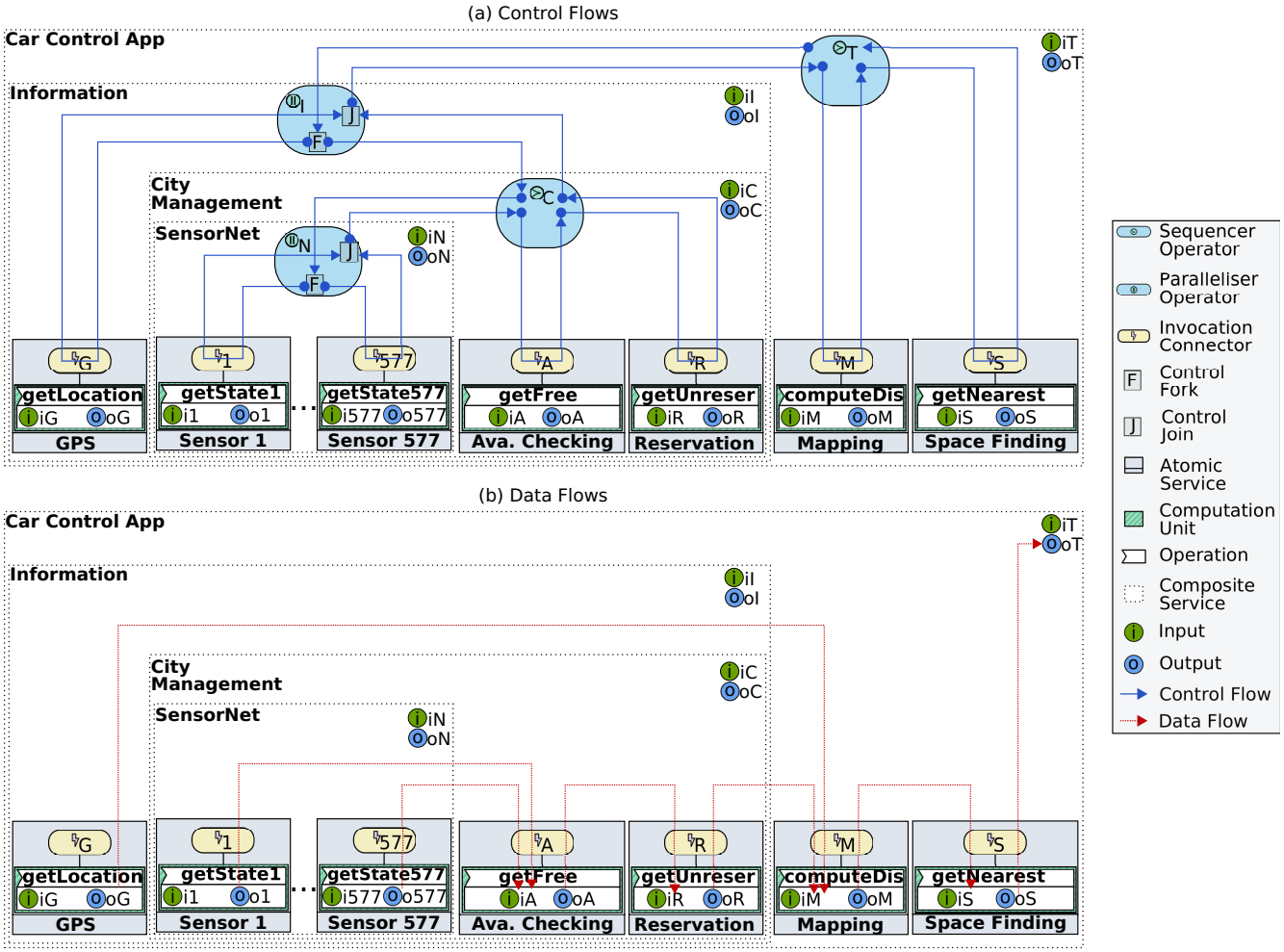


FIGURE 19: Orthogonal control flows and data flows in the smart parking scenario.

control generally travels faster than data over the network, and they may happen even in sequential workflows. This implies that an operation can be executing while control waits in the invocation connector of the next service in a pipeline.

6. DISTRIBUTED DATA FLOWS VS DECENTRALISED DATA FLOWS IN THE SMART PARKING SCENARIO

This section presents a quantitative comparison between the proposed approach (when data is separated from control) and distributed data flows (when data follows control), in terms of network traffic generated and response time perceived. Our evaluation is based on two OMNET++ simulations in which composition operators and invocation connectors are OMNET++ modules, whereas control flow and data flow are channels between modules.[8] For fairness, we assume ideal wireless channels communicating over IEEE 802.11a links at a rate of 54Mbps, with no delay and no bit error rate. The configuration of our simulations is shown in Fig. 20.

An OMNET++ experiment simulates drivers using the smart parking application between 07:59:45 and 16:26:47 (see Appendix A). In all the experiments, we assume that there is no network traffic for invoking operations, since invocation connectors reside in the same device as their connected operations. Also, composition operators and invocation connectors do not incur any processing time since they only forward data or control. The latter is just a 1-byte signal that activates an operator or a connector.

For simplicity and clarity, we also assume that IoT devices hosting atomic services have a fixed data processing time of 100μ seconds. This assumption is valid since having equal data processing rates does not influence the network traffic generated by atomic services.

Considering the above assumptions, the first experiment simulates decentralised data flows in the smart parking system. Here, control is exchanged among composition operators and invocation connectors, while data is passed between producer connectors and consumer connectors via ideal OMNET++ channels (see Fig. 20). These channels simulate read-write operations on a decentralised space (regardless its implementation).

[8] https://omnetpp.org/

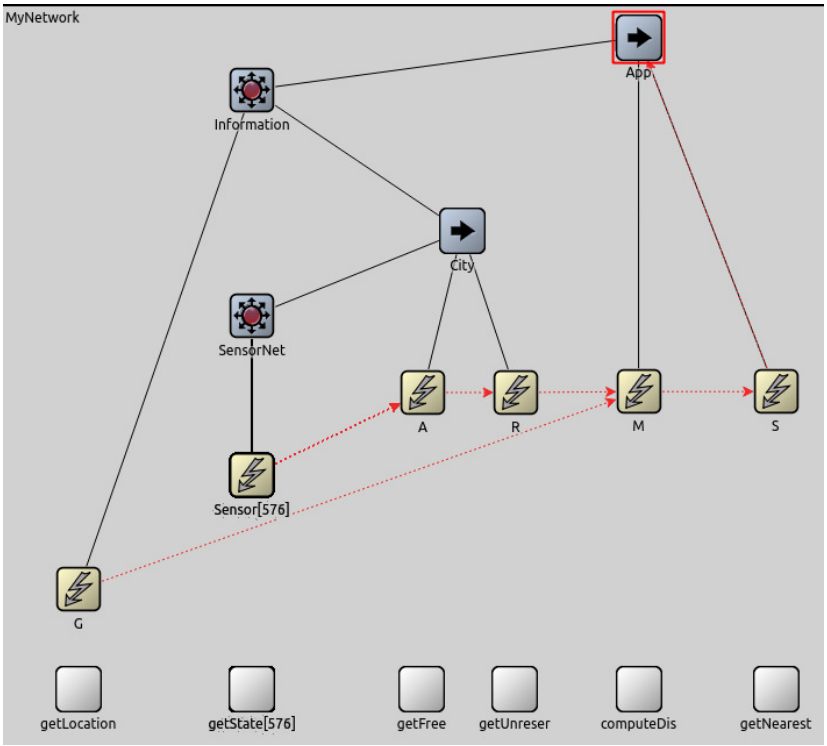


FIGURE 20: OMNET++ configuration for the smart parking system. The simulation on distributed data flows do not use the channels between invocation connectors, whereas the simulation on decentralised data flows use all visible channels.

TABLE 2: Network traffic in the smart parking scenario (from 07:59:45 to 16:26:47).

	Data Traffic (MB)	Control Traffic (MB)	Total (MB)
Decentralised	51.49	0.44	51.93
Distributed	140.41	0.22	140.63

These actions are performed by invocation connectors on behalf of their connected operations. In this experiment, the total network traffic from 07:59:45 to 16:26:47 is 51.93MB (see Table 2). The additional traffic with respect to the calculation described in Appendix B corresponds to the bytes required for control exchange (i.e., 0.44MB).[9] This extra traffic is minimal since control is just a 1-byte signal that does not carry any additional data.

The second simulation considers a distributed version of the smart parking scenario, in which data passes alongside control through invocation connectors and composition operators. The only exception is when control traverses from $\circ_T$ to G, and from $\circ_I$ to the invocation connectors 1 and 577 (see Fig. 21(b)). This is because the *GPS Service* and the *Sensor Services* do not require any input data from the *CarControlApp* composite (see Fig. 18(a)). Using distributed data flows, the total network traffic from 07:59:45 to 16:26:47 is 140.63MB (see Table 2).

Although our approach requires 50% more control messages than its counterpart, the network traffic is dramatically minimised by a rate of 2.71 times, i.e., $(1 - \frac{51.93}{140.63}) \cdot 100 \approx 63.07\%$ less network traffic than the distributed version. Particularly, the main benefit occurs when data is passed directly from the *Reservation Service* to the *Mapping Service*, since the distributed approach requires four network hops whereas our approach requires only one (see Fig. 21).

Reducing network traffic improves the overall response time of the smart parking system by an average factor of 4. This factor is computed as follows:

$$\frac{\sum_{i=1}^{373} \frac{\psi_i^{\alpha}}{\psi_i^{\beta}}}{373} \tag{3}$$

where ψ_i^{α} is the response time perceived by driver i when requesting a parking space under distributed data flows and ψ_i^{β} is the response time perceived by driver i when requesting a parking space under decentralised data flows. The response times (in seconds) are individually collected from our simulations for the 373 drivers arriving between 07:59:45 and 16:26:47.[10]

As an example, Fig. 21 shows the time window for the driver's request at 08:26 (i.e., 1621 seconds after 07:59:45), where it is clear that the system is executed in 0.08 seconds using the distributed approach

[9] For increased clarity in our motivation example (see Section 2), the calculations in Appendix B do not consider control flow.

[10] Appendix A presents the computation for the number of drivers.

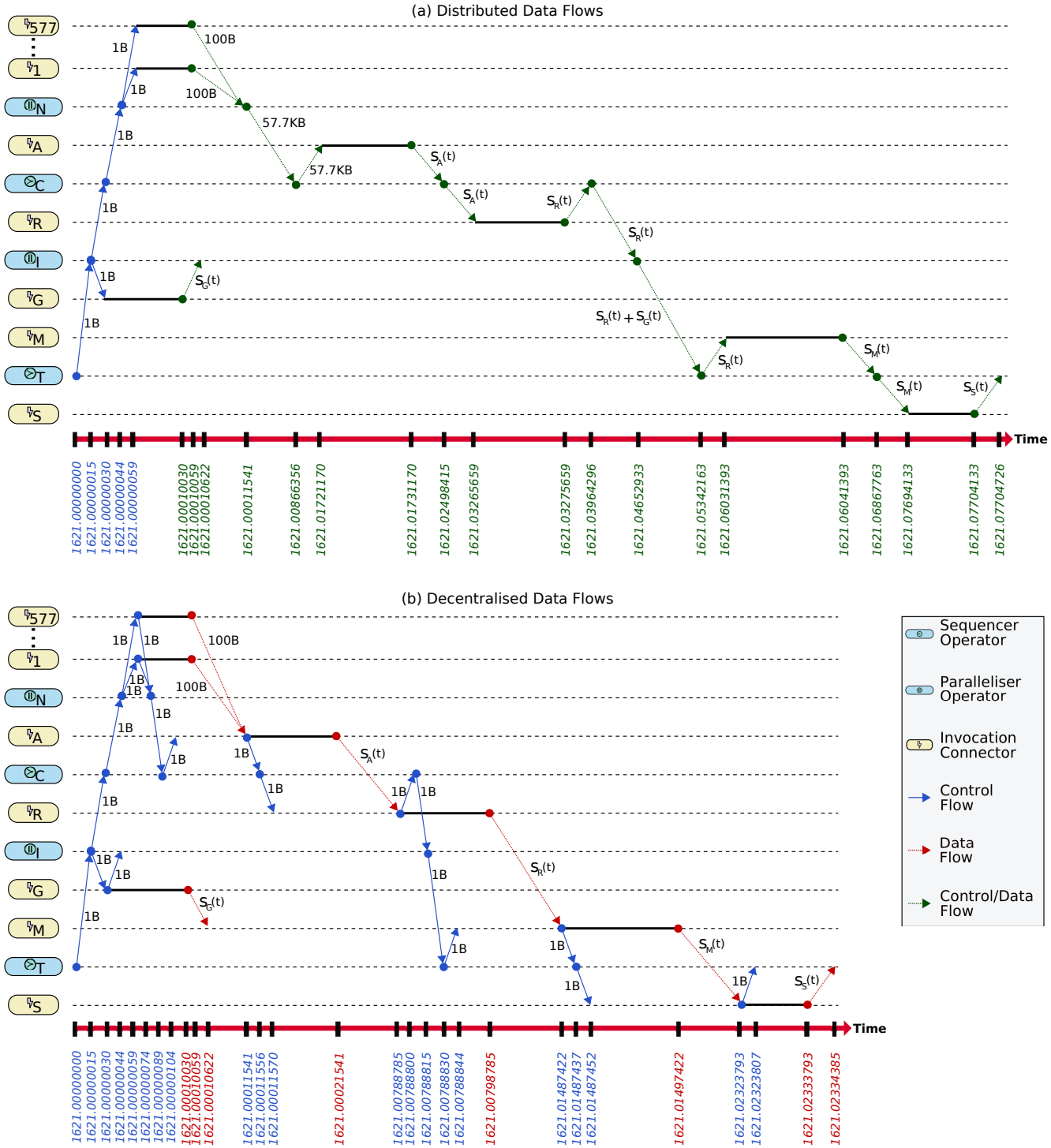
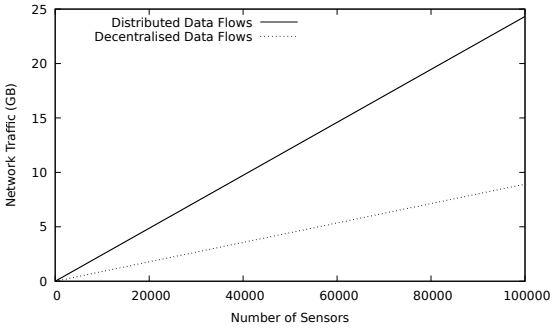


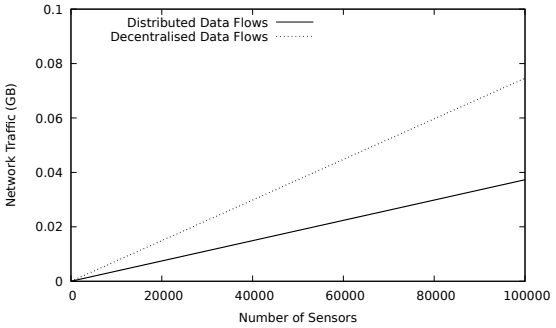
FIGURE 21: Decentralised data flows vs distributed data flows in the smart parking scenario.

and in 0.02 seconds with the decentralised version. This improvement is due to the fact that data and control travel independently over the network when decentralised data flows are used, so that control arrives before data in certain cases (e.g., see the timeline of R) and data arrives before control in others (e.g., see the timeline of M).

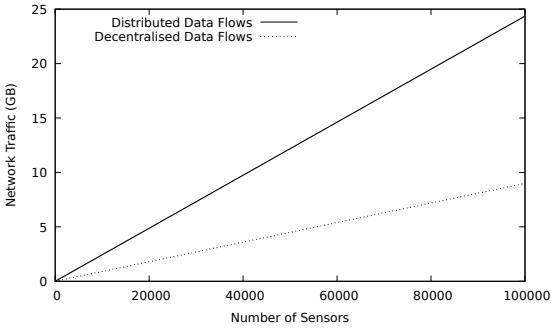
Table 2 presents the results of a quantitative evaluation that considers a few services. However, the benefit of our approach becomes more evident as the number of services increases. To evaluate this property, known as *functional scalability*, we conducted an experiment in which we gradually increase the data being processed by the smart parking application through the continuous addition of sensor services (up to 100,000). The aim of this experiment is to analyse network traffic in the order of Gigabytes. The results are shown in Fig. 22.



(a) Data traffic.



(b) Control traffic.



(c) Total network traffic.

FIGURE 22: Impact of increasing the number of sensor services in the smart parking scenario.

Fig. 22 shows that our approach outperforms distributed data flows in terms of data traffic, whereas the distributed version outperforms our approach in terms of control traffic. Despite this, the total network traffic is linearly improved by our approach since control messages are minimal (i.e., they always oscillate below 0.08 GB). Consequently, the total traffic and the data traffic are almost identical (cf., Fig. 22(a) and Fig. 22(c)).

The next section presents a generic evaluation of functional scalability that is independent of any implementation or specific case studies. In this evaluation, we do not only increase services horizontally (for the same composite) but also vertically (by increasing the number of composition levels).

7. EVALUATION OF FUNCTIONAL SCALABILITY

This section presents a comparative evaluation between decentralised data flows (when control flow and data flow are separated – Fig. 8(b)) and distributed data flows (when data follows control – Fig. 8(a)), in terms of functional scalability. To do so, two major research questions are studied: (RQ1) Does the proposed approach scale with the number of services? and (RQ2) Under which conditions are decentralised data flows beneficial?

To answer the above questions, we conducted a series of experiments based on the following statements.

Definition 7.1. *$G_\alpha = (V_\alpha, E_\alpha, \Omega_\alpha)$ is a weighted graph of distributed data flows in which V_α is a set of parameters, E_α is a set of directed data flows (between parameters) and $\Omega_\alpha : E_\alpha \to \mathbb{R}_{>0}$ is a function that maps a distributed data flow to a network communication cost s.t. $V_\alpha \subset (V_p \cup V_c)$.*

Remark 1. V_p is the type of a producer parameter and V_c is the type of a consumer parameter.

Definition 7.2. *$M = \langle e_1, \ldots, e_l \rangle$ is the type of a finite walk in G_α for moving a data value through l data flows, from a producer parameter $v_1 \in V_p$ to a consumer parameter $v_{l+1} \in V_c$ s.t. $e_{j\in[1,l]} \in E_\alpha$ and $v_{j\in[1,l+1]} \in V_\alpha$, $\forall p \in (V_\alpha \cap V_p)\ \exists\ M$.*

Definition 7.3. *Given G_α, $\alpha : M \to \mathbb{R}_{>0}$ is the function that computes the total network cost of routing data from a producer parameter p to a consumer parameter q, via distributed data flows. The function α is given as follows:*

$$\alpha(M') = \sum_{j=1}^{|E(M')|} \Omega_\alpha(e_j)$$

where $M' \in M$, $e_{j\in[1,|E(M')|]} \in E(M')$, $E(M') \subseteq E_\alpha$, $p \in (V_\alpha \cap V_p)$ and $q \in (V_\alpha \cap V_c)$.

Notation 8. Let $\omega : V_p \to \mathbb{R}_{>0}$ be the function that computes the network cost of writing/producing a parameter value on a data space.

Notation 9. Let $\Gamma : V_c \to \mathbb{R}_{>0}$ be the function that computes the network cost of reading/consuming a parameter value from a data space.

Definition 7.4. *$G_\beta = (V_\beta, E_\beta, \Omega_\beta)$ is a weighted graph of decentralised data flows in which V_β is a set of parameters, E_β is a set of directed data flows (between parameters) and $\Omega_\beta : E_\beta \to \mathbb{R}_{>0}$ is a function that maps a decentralised data flow to a network communication cost s.t. $V_\beta \subseteq (V_p \cup V_c)$. The function Ω_β is defined as follows:*

$$\Omega_\beta(e) = (\omega \circ \Pi_1^\beta)(e) + (\Gamma \circ \Pi_2^\beta)(e)$$

where $e \in E_\beta$, $\Pi_1^\beta(e) \in (V_\beta \cap V_p)$ and $\Pi_2^\beta(e) \in (V_\beta \cap V_c)$.

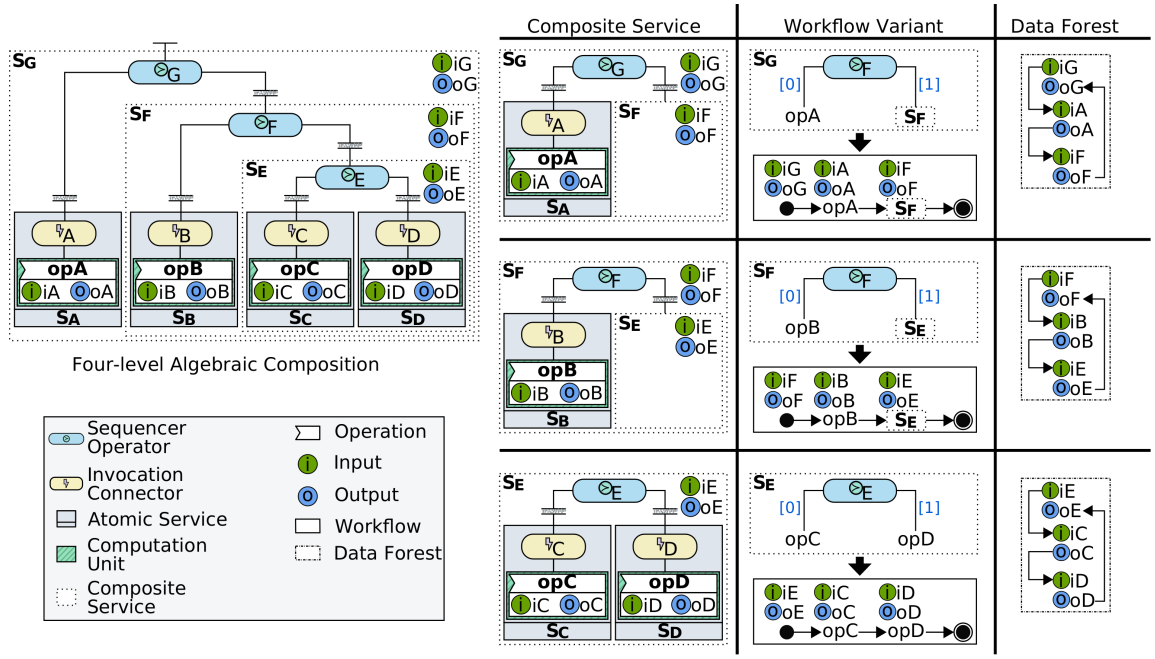


FIGURE 23: Initial experimental setting.

Remark 2. The functions $\Pi_1^\beta : E_\beta \rightarrow V_\beta$ and $\Pi_2^\beta : E_\beta \rightarrow V_\beta$ are the tail map and the head map of an edge in G_β, respectively.

Definition 7.5. *Given G_β, $\beta : E_\beta \rightarrow \mathbb{R}_{>0}$ is the total network cost of routing data from a producer parameter p to a consumer parameter q, via a decentralised data flow. The function β is notably given as follows:*

$$\beta(e) = \Omega_\beta(e)$$

where $e \in E_\beta$, $p = \Pi_1^\beta(e)$, $q = \Pi_2^\beta(e)$, $p \in (V_\beta \cap V_p)$ and $q \in (V_\beta \cap V_c)$.

We assume that Ω_α, Γ and ω include all network communication factors, such as marshalling, unmarshalling, latency and bandwidth, just to name a few.

7.1. Initial Experimental Setting

The worst performance of a distributed approach occurs when all composite services are sequential. So, for our evaluation we consider the DX-MAN composition shown in Fig. 23, which has four hierarchy levels, four atomic services and three composites. The sequential workflow variants selected for each composite and their respective data forests are also presented in the figure.

Fig. 24 shows the resulting workflow control flow and the data exchange approaches that we use in our experiments. For the distributed approach (see Fig. 24(b)), we consider the graph G_α in which data follows control and passes through workflow parameters via nine data flow edges with a $\Omega_\alpha(e_j)$ cost each s.t. $e_{j\in[1,9]} \in E_\alpha$. Particularly, G_α has five data flow walks since the number of producer parameters is directly

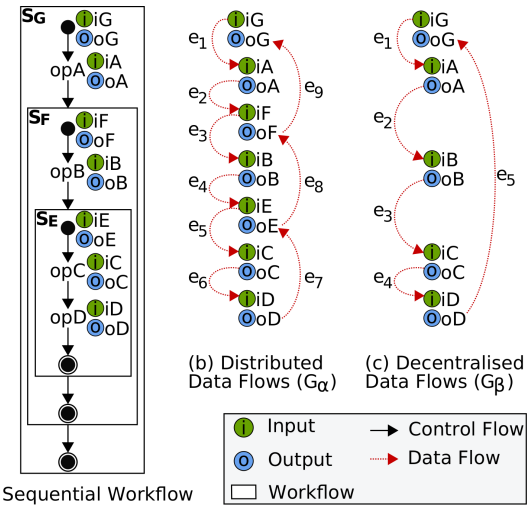


FIGURE 24: Distributed data flows vs decentralised data flows.

proportional to the number of walks: $M_{iG} = \langle e_1 \rangle \in M$, $M_{oA} = \langle e_2, e_3 \rangle \in M$, $M_{oB} = \langle e_4, e_5 \rangle \in M$, $M_{oC} = \langle e_6 \rangle \in M$ and $M_{oD} = \langle e_7, e_8, e_9 \rangle \in M$ s.t. $iG, oA, oB, oC, oD \in (V_\alpha \cap V_p)$ and $e_{j\in[1,9]} \in E_\alpha$.

For the decentralised approach (see Fig. 24(c)), we consider the graph G_β in which data is separated from control and passes from producer parameters to consumer parameters directly. In this case, there are five data flows each with a network cost of $\Omega_\beta(e_j)$ s.t. $e_{j\in[1,5]} \in E_\beta$. For example, the cost of routing the data produced by oA is $\Omega_\beta((oA, iB)) = \omega(oA) + \Gamma(iB)$. As G_β is constructed with Algorithm 1 and does not consider any data exchanges through workflow parameters, the number of edges is directly proportional

to the number of producer parameters (i.e., $|E_\beta| = |V_\beta \cap V_p| = 5$) and the number of vertices is twice the number of producers (i.e., $|V_\beta| = 2|V_\beta \cap V_p| = 10$). Therefore, $|V_\beta| < |V_\alpha| \ \wedge \ |E_\beta| < |E_\alpha| \because |V_\alpha| = 14 \ \wedge \ |E_\alpha| = 9$.

7.2. RQ1: Does the proposed approach scale with the number of services?

We conducted an experiment to dynamically increase the number of composites in the (four-level) composition shown in Fig. 23. The experiment is carried out in $\tau = 200000$ steps with $|E_\alpha| \in (\mathbb{N} \cap [9, f(\tau)])$ and $|E_\beta| \in (\mathbb{N} \cap [5, g(\tau)])$ s.t. $f(c) = 4c + 9$ is the total number of distributed data flows and $g(c) = 2c + 5$ is the total number of decentralised data flows, when there are $c \in (\mathbb{N} \cap [0, \tau])$ new composites in S_F.

At each step of the experiment, we compose a replica of S_E into S_F and define a sequential workflow variant for the invocation of the operations *opC* and *opD* (see Fig. 23). At the end of the experiment, there is a data pipeline in G_α for $\tau + 1$ composites in S_F. In the case of G_β, there are additional data flows between the parameters of the atomic services added.

As they capture the growth pattern of data flows, we use the functions $f(c)$ and $g(c)$ to analytically compare $|E_\alpha|$ and $|E_\beta|$ as the number of composites increases. Fig. 25 shows the result of our analysis under the assumption that the weight functions of E_α and E_β are $\Omega_\alpha : E_\alpha \to \{1\}$ and $\Omega_\beta : E_\beta \to \{1\}$. On that basis, it is clear that the number of data exchanges grows linearly with the number of composite services and that the number of distributed data flows grows twice faster than its counterpart. The latter can be strictly explained by the average rate of change $\frac{\Delta f}{\Delta g} = 2$ in the interval $[0, \tau]$.

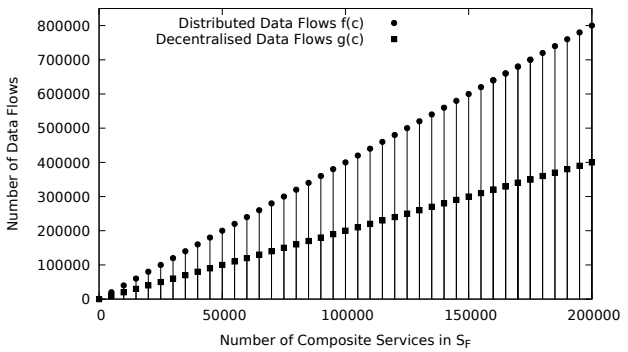


FIGURE 25: Impact of data flows when increasing the number of composite services in S_F.

7.3. RQ2: Under which conditions are decentralised data flows beneficial?

We conducted two experiments to determine if there is a benefit of decentralised data flows as the number of levels increases in a DX-MAN composition. To do so, we consider the total network costs of passing the output oD (of the atomic service S_D) to an output oT (of a top-level composite), using both the distributed approach and the decentralised one s.t. $oT \in V_c$. For both experiments, we assume that $im(\Omega_\alpha), im(\omega), im(\Gamma) \in \mathbb{R} \cap (0, 1]$.

For each experiment, we compute ten samples over 100 levels each. At each sample, we increase the number of composition levels by 1 and define random network costs for E_α and E_β using Ω_α and Ω_β, respectively. The decentralised approach yields a graph with the same number of edges no matter the composition levels, whereas the distributed approach increases the number of edges by 1 at every level. Thus, $|M_{oD}|$ also increases by 1. The improvement rate of the decentralised approach is given by $1 - \frac{\beta((oD, oT))}{\alpha(M_{oD})}$.

Fig. 26 shows the result of our experiments, where it is clear that the benefit of decentralised data flows is logarithmic and, as such, it becomes more evident as the number of levels of a DX-MAN composition increases. This is because $|M_{oD}|$ increases at every level and so $\alpha(M_{oD})$. In the experiments, we allocate network costs using random values uniformly distributed in $(0, 1] \in \mathbb{R}$. However, if the cost of performing operations on a data space (i.e., $\beta((oD, oT))$) is more expensive than the total cost of routing data through composite coordinators (i.e., $\alpha(M_{oD})$), the decentralised approach is outperformed by the distributed one. Thus, a DX-MAN composition only benefits from the proposed approach iff $\beta((oD, oT)) < \alpha(M_{oD})$.

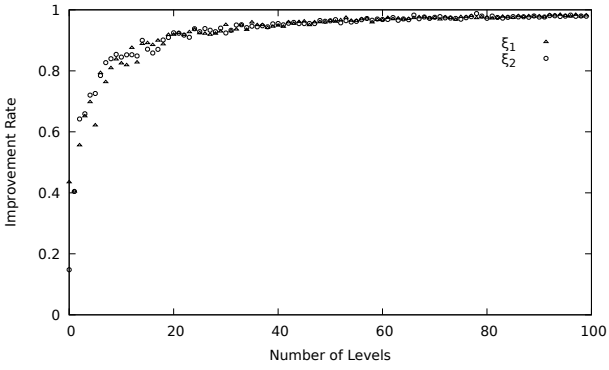


FIGURE 26: Improvement rate of decentralised data flows when increasing the number of levels in a DX-MAN composition: ξ_1 and ξ_2 correspond to Experiment 1 and Experiment 2, respectively.

8. CONCLUSIONS

In this paper, we presented a functionally scalable approach that semantically separates control and data in DX-MAN compositions for the realisation of decentralised data flows in service-oriented IoT systems. At design-time, the algebraic semantics of the model allows the definition of data forests which denote data dependencies between service operation parameters. At deployment-time, data forests are (manually) refined and

(automatically) analysed by an algorithm that leverages the separation of control and data. The algorithm works for any workflow in a composite workflow space, and produces a direct mapping from consumer parameters to producer parameters. Thus, preventing coordinators (i.e., exogenous operators) from passing data at run-time. In fact, exogenous operators coordinate an IoT system execution by passing control only. To realise our approach, we implemented it on top of the DX-MAN platform which uses the Blockchain as the underlying data space for persisting direct mappings and managing data at run-time. It is important to mention that our approach is not exclusive to a specific data space implementation (e.g., Blockchain or OpenLink Data Spaces). For that reason, we evaluated the benefits of the proposed approach rather than the benefits of a particular implementation.

Unfortunately, as other composition models have their own constructs and do not define composition operators like DX-MAN, it is not possible to extend other composition model semantics using our proposed approach. Instead, a custom extension per model would be required to enable the semantic separation of data and control. Another interesting future direction is to explore the possibility of incorporating stateful services into the semantics of DX-MAN.

DX-MAN is a service composition model that semantically separates data, control and computation for separate reasoning, monitoring, maintenance and evolution of such dimensions. More concretely, this separation allows passing data from producer parameters to consumer parameters directly, and enables the use of distinct technologies to manage control flows and data flows separately. For example, in our implementation we use CoAP to pass control between exogenous operators and the Blockchain for handling data flows.

Our experimental results confirm that our approach scales well with the number of services, by reducing the number of data flows by an average factor of two with respect to a distributed approach (where data follows control). They also show that our approach scales well with the number of levels of a DX-MAN composition. From our results, we found that the proposed approach provides the best performance when the cost of performing operations on a data space is less than the cost of exchanging data over the network. Thus, our solution is potentially beneficial for large-scale IoT systems in which loosely-coupled services exchange huge amounts of data continuously.

APPENDIX A

This appendix presents the notation required for the analysis of the IoT scenario described in Section 2. This scenario considers a real-world data set publicly available at `https://archive.ics.uci.edu/ml/datasets/Parking+Birmingham#` which collects occupancy status in car parks operated by the Birmingham City Council, from 07:59:45 to 16:26:47 on October 22, 2016 [52]. Table A.1 presents the data obtained for the car park *BHMBCCMKT01* whose total capacity is 577. It particularly shows the measurement time, the t seconds elapsed after 07:59:45, the number of occupied spaces $O(t)$ and the rate of available parking spaces (calculated by $1 - \frac{O(t)}{577}$).

TABLE A.1: Occupancy in the car park BHMBCCMKT01.

Measurement Time	t	$O(t)$	Availability Rate
07:59:45	0	57	0.90
08:26:46	1621	59	0.90
08:59:44	3599	72	0.88
09:26:47	5222	93	0.84
09:59:44	7199	130	0.77
10:26:47	8822	156	0.73
10:59:47	10802	198	0.66
11:32:43	12778	239	0.59
11:59:46	14401	261	0.55
12:32:47	16382	324	0.44
12:59:47	18002	359	0.38
13:32:43	19978	380	0.34
13:59:48	21603	408	0.29
14:39:47	24002	430	0.25
14:59:48	25203	424	0.27
15:26:49	26824	403	0.30
15:59:47	28802	384	0.33
16:26:47	30422	340	0.41

Notation 10. Let $T = \{0, 1621, 3599, 5222, 7199, 8822, 10802, 12778, 14401, 16382, 18002, 19978, 21603, 24002, 25203, 26824, 28802, 30422\}$ be the set of measurement times in our sample. Then, $\Delta O(t_i, t_{i+1})$ is the function that computes the number of drivers arriving to the car park *BHMBCCMKT01* between t_i and t_{i+1} seconds:

$$\Delta O(t_i, t_{i+1}) = \begin{cases} O(t_{i+1}) - O(t_i) & O(t_{i+1}) > O(t_i) \\ 0 & otherwise \end{cases} \tag{4}$$

s.t. $\sum_{i=1}^{|T|-1} \Delta O(t_i, t_{i+1}) = 373$ and $t_i, t_{i+1} \in T$. This means that there are 373 drivers arriving to the car park *BHMBCCMKT01* between 07:59:45 and 16:26:47.

Considering the columns 2 and 4 from our data set, we performed a cubic regression to approximate a function $A(t)$ for the rate of available parking spaces at t seconds. The resulting equation has an average relative error of 1.98% and it is defined as follows:

$$A(t) = 0.00000000000009516t^3 - 0.0000000363813682t^2 + 0.00000641771909518t + 0.89631130855805541$$

where $0 \leq t \leq 30422$.

The probability of a parking space being unreserved at t seconds is then $A(t)$. Following the same trend of demand for parking spaces, the rate of parking spaces that are both free and unreserved at t seconds is given by the function $U(t)$ as follows:

$$U(t) = A(t)^2$$

where $0 \leq t \leq 30422$.

APPENDIX B

This appendix presents the result of the calculations for the total data transmitted over the network in our IoT scenario (see Section 2 and Appendix A). The calculations are presented for each data exchange approach, i.e., centralised data flows, distributed data flows and decentralised data flows. For all of them, we assume that:

1. There are 577 occupancy sensors near a driver, corresponding to each space in the car park *BHMBCCMKT01*.
2. Each *Sensor Service* is hosted in an occupancy sensor and produces 100 bytes.
3. Once all sensor data is collected by a reducer, it is processed by the *Availability Checking Service*.
4. The *Mapping Service* appends 20 bytes to each parking space that is both free and unreserved. These extra bytes correspond to the distance between the driver's location and each occupancy sensor.
5. The *Space Finding Service* produces 40 bytes for the coordinates of the nearest parking space.
6. There is a driver requesting a nearest parking space every $\frac{t_{i+1}-t_i}{\Delta O(t_i,t_{i+1})}$ seconds between $t_i \in T$ and $t_{i+1} \in T$ (see Notation 10 and Equation 4).
7. The *GPS* service produces 40 bytes for the coordinates of the driver.
8. The driver always finds a nearest parking space, since the car park is never full (see $O(t)$ in Table A.1).

Notation 11. Considering assumptions (1) and (2), let $\theta = 577 \cdot 100 = 57700$ be the constant for the total sensor data (in bytes) produced for a driver's request.

For the sake of clarity and conciseness, we use the functions below to compute the number of bytes returned by the *Availability Checking Service*, the *Reservation Service*, the *Mapping Service* and the *Space Finding Service*, respectively.

Notation 12. Let $S_A(t) = \theta \cdot A(t)$ be the function that computes the number of bytes returned by the *Availability Checking Service*. These bytes correspond to the list of parking spaces that are available t seconds after 07:59:45 s.t. $0 \leq t \leq 30422$.

Notation 13. Let $S_R(t) = \theta \cdot U(t)$ be the function that computes the number of bytes returned by the *Reservation Service*. These bytes correspond to the list of parking spaces that are both available and unreserved t seconds after 07:59:45 s.t. $0 \leq t \leq 30422$.

Notation 14. Considering assumptions (1) and (4), let $S_M(t) = S_R(t) + 577 \cdot U(t) \cdot 20$ be the function that computes the number of bytes returned by the *Mapping Service*, where $0 \leq t \leq 30422$. These bytes correspond to the list of parking spaces that are both available and unreserved t seconds after 07:59:45, plus the appended bytes for the distance between each parking space in the list and the driver's location.

Notation 15. Considering the assumption (5), let $S_S = 40$ be the constant for the number of bytes returned by the *Space Finding Service*, which represent the coordinates of the nearest parking space.

Notation 16. Considering the assumption (7), let $S_G = 40$ be the constant for the number of bytes returned by the *GPS Service*, which represent the coordinates of the driver.

A centralised approach for exchanging data requires two data flows for moving data from a service producer to a service consumer. Therefore, B_γ denotes the total data exchanged over the network from 07:59:45 to 16:26:47 when a centralised approach is used:

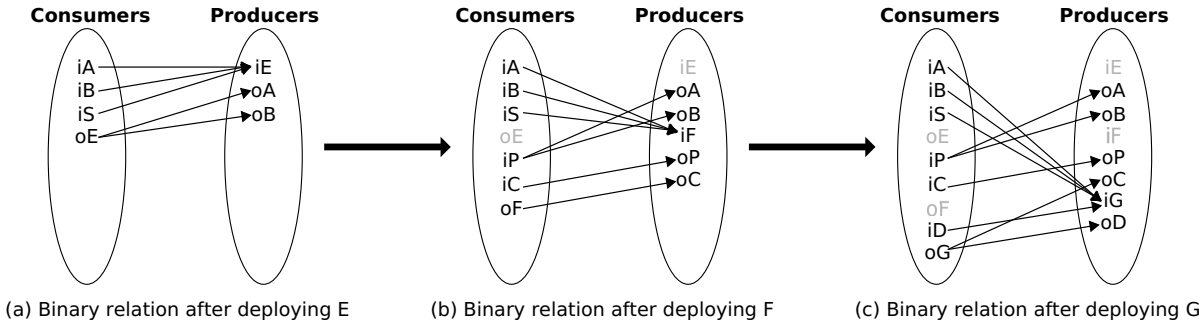
$$B_\gamma = \sum_{i=1}^{|T|-1} \sum_{j=1}^{\Delta O(t_i,t_{i+1})} 2S_G+2\theta+2S_A(t_i+\frac{t_{i+1}-t_i}{\Delta O(t_i,t_{i+1})}j) +2S_R(t_i+\frac{t_{i+1}-t_i}{\Delta O(t_i,t_{i+1})}j)+2S_M(t_i+\frac{t_{i+1}-t_i}{\Delta O(t_i,t_{i+1})}j)+2S_S \approx 102.98MB \quad (5)$$

For the distributed approach, each coordinator requires two data flows for passing data between a service producer and a service consumer. In addition, *Coordinator1* requires a data flow for moving data to the *Coordinator2* which, in turn, requires another data flow for passing data to the *Coordinator3*. Thus, the exchanges between the *Availability Checking Service* and the *Reservation Service* are done with three data flows. Likewise, there are three data flows for passing data from the *Reservation Service* to the *Mapping Service*. The symbol B_α denotes the total data exchanged over the network from 07:59:45 to 16:26:47 when a distributed approach is used:

$$B_\alpha = \sum_{i=1}^{|T|-1} \sum_{j=1}^{\Delta O(t_i,t_{i+1})} 2S_G+2\theta+3S_A(t_i+\frac{t_{i+1}-t_i}{\Delta O(t_i,t_{i+1})}j) +3S_R(t_i+\frac{t_{i+1}-t_i}{\Delta O(t_i,t_{i+1})}j)+2S_M(t_i+\frac{t_{i+1}-t_i}{\Delta O(t_i,t_{i+1})}j)+2S_S \approx 123.36MB \quad (6)$$

A decentralised approach is the most efficient since it requires only one data flow for moving data from a service producer to a service consumer. Based on this premise, B_β denotes the total data exchanged over the network from 07:59:45 to 16:26:47 when a decentralised approach is used:

$$B_\beta = \sum_{i=1}^{|T|-1} \sum_{j=1}^{\Delta O(t_i,t_{i+1})} S_G+\theta+S_A(t_i+\frac{t_{i+1}-t_i}{\Delta O(t_i,t_{i+1})}j) + S_R(t_i+\frac{t_{i+1}-t_i}{\Delta O(t_i,t_{i+1})}j)+S_M(t_i+\frac{t_{i+1}-t_i}{\Delta O(t_i,t_{i+1})}j)+S_S \approx 51.49MB \quad (7)$$



(a) Binary relation after deploying E (b) Binary relation after deploying F (c) Binary relation after deploying G

FIGURE 27: Deployment of the data forest shown in Fig. 28.

Taking into account the above calculations, the distributed approach leads to $(\frac{B_\alpha}{B_\gamma} - 1) \cdot 100 \approx 19.80\%$ more network traffic than the centralised approach. Likewise, the decentralised approach produces $(1 - \frac{B_\beta}{B_\gamma}) \cdot 100 \approx 50.00\%$ less network traffic than the centralised approach and $(1 - \frac{B_\beta}{B_\alpha}) \cdot 100 \approx 58.26\%$ less traffic than the distributed one.

APPENDIX C

This appendix presents an example of how data forests are analysed in a bottom-up fashion using the Algorithm 1 at deployment-time. Our example considers the three-level data forest depicted in Fig. 28, which obeys the connection rules described in Section 3.2. Here, G is the forest of a parallel workflow that simultaneously executes the corresponding workflow for F and an atomic operation (with input iD and output oD). The workflow for F sequentially executes the workflow for E and then an atomic operation (with input iC and output oC). To pre-process data, F has a filter that takes the data produced by E before sending it to the corresponding operation. In our example, E is the lowest-level data forest defined by a branchial workflow that uses the input parameter iS to choose an operation to execute (out of two possible ones).

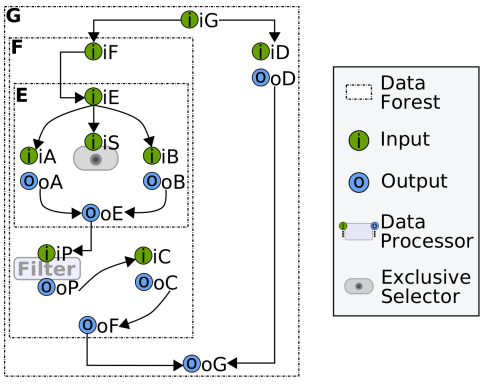


FIGURE 28: Three-level data forest.

As described in Section 3.4, the deployment process begins at the bottom-level. So, we first analyse the edges of the branchial data forest E via the Algorithm 1. The resulting binary relation is shown in Fig. 27(a).

In the next step, we analyse the edges of the sequential data forest F to yield the binary relation described in Fig. 27(b). Here, we can notice that the relations from the output oE are now the relations from iP. This is because the filter input does not require to read from oE. Instead, the filter reads data directly from any of the actual producers, viz. the operation output oA or the operation output oB. Likewise, the input parameters iA, iB and iS do not read data from iE since a new (intermediate) top-level data producer has been found (i.e., iF). Complementarily, the operation input iC reads data directly from the filter output, while the (intermediate) top-level output oF reads data from oC.

Finally, in the last deployment step, the edges of the data forest G are analysed to yield the binary relation depicted in Fig. 27(c). As iG is the new top-level data producer, the relations from consumers iA, iB and iS are updated accordingly. Similarly, as G is a parallel forest, the top-level output oG takes both the data source from oF (i.e., oC) and the output oD. At the end of the deployment process, we have a mapping scheme that discards unnecessary relations to allow data consumers read values directly from data producers.

APPENDIX D

Fig. 29 presents the JavaScript source code of the three smart contracts used in our implementation. The source code follows the contract model definition described in Section 4.

```
/**
 * @param {com.dxman.blockchain.CreateParameters} tx
 * @transaction
 */
async function createParameters(tx) {
  var resources = [];
  let reg = await getAssetRegistry("com.dxman.blockchain.
       Parameter");
  for(i = 0; i < tx.parameters.length; i++) {
    let resource = getFactory().newResource("com.dxman.
         blockchain", "Parameter", tx.parameters[i].id);
    resource.parameterId = tx.parameters[i].parameterId;
    resource.workflowId = tx.parameters[i].workflowId;
    resource.value = tx.parameters[i].value;
    resource.timestamp = tx.timestamp;
    resource.updater = tx.parameters[i].updater;
    resource.producers = tx.parameters[i].producers;
    resources[i] = await reg.add(resource);
  }
}
/**
 * @param {com.dxman.blockchain.ReadParameters} tx
 * @returns {string[]}
 * @transaction
 */
async function readParameters(tx) {
  var parameters = [];
  for(let i = 0; i < tx.parameters.length; i++) {
    parameters[i] = readPar(tx.parameters[i], tx.
         workflowTimestamp);
  }
  return await Promise.all(parameters).then(values => {return
       values;},
    error => {return [];});
}
function readPar(parameter, workflowTimestamp) {
  if(parameter.producers.length > 0) {
    for(i = 0; i < parameter.producers.length; i++) {
      if(parameter.producers[i].timestamp >= workflowTimestamp &&
        (parameter.producers[i].updater == parameter.producers[i].
             id ||
        parameter.producers[i].updater == "user")) {
        return Promise.resolve(parameter.producers[i].value);
      }
    }
    return Promise.reject('PARAMETER_VALUE_NOT_FOUND');
  } else return Promise.reject('NO_PRODUCERS');
}
/**
 * @param {com.dxman.blockchain.UpdateParameters} tx
 * @transaction
 */
async function updateParameters(tx) {
  let paramsArray = [];
  for(i = 0; i < tx.updates.length; i++) {
    tx.updates[i].parameter.timestamp = tx.timestamp;
    tx.updates[i].parameter.value = tx.updates[i].newValue;
    tx.updates[i].parameter.updater = tx.updates[i].updater;
    paramsArray[i] = tx.updates[i].parameter;
    let updateEvent = getFactory().newEvent('com.dxman.
         blockchain', 'UpdateParameterEvent');
    updateEvent.parameter = tx.updates[i].parameter.id;
    updateEvent.newValue = tx.updates[i].parameter.value;
    updateEvent.updater = tx.updates[i].updater;
    emit(updateEvent);
  }
  let reg = await getAssetRegistry("com.dxman.blockchain.
       Parameter");
  await reg.updateAll(paramsArray);
}
```

FIGURE 29: Smart contracts logic.

APPENDIX E

This appendix presents the notation used throughout the paper.

E.1 IoT Scenario

$O(t)$	Number of occupied parking spaces at t seconds after 07:59:45
T	Set of measurement times (in seconds)
$\Delta O(t_i, t_{i+1})$	Number of drivers requesting a nearest space between $t_i \in T$ and $t_{i+1} \in T$ seconds
$A(t)$	Rate of free spaces at t seconds after 07:59:45
$U(t)$	Rate of spaces that are both free and unreserved at t seconds after 07:59:45
θ	Total sensor data (in bytes) produced for a driver's request
$S_A(t)$	Number of bytes produced by the *Availability Checking Service* for a driver's request
$S_R(t)$	Number of bytes produced by the *Reservation Service* for a driver's request
$S_M(t)$	Number of bytes produced by the *Mapping Service* for a driver's request
S_S	Fixed number of bytes produced by the *Space Finding Service*
S_G	Fixed number of bytes produced by the *GPS Service*
B_γ	Number of bytes exchanged in our scenario using a centralised approach
B_α	Number of bytes exchanged in our scenario using a distributed approach
B_β	Number of bytes exchanged in our scenario using a decentralised approach

E.2 The DX-MAN Model

$\mathbb{S}$	The type of a service
$S \in \mathbb{S}$	A service S
$\mathbb{W}$	The type a workflow space
$W \in \mathbb{S}$	A workflow space W
$\circ$	The type of a composition operator
$\circ_Z$	A composition operator defining a composite service Z
$\mathbb{D}$	The type of a parameter
F	A data forest F
$V(F)$	The set of parameter vertices in a data forest F
$v \in V(F)$	A parameter v in a data forest F
$E(F)$	The set of edges in a data forest F
$e \in E(F)$	An edge e (i.e., a pair of parameters) in a data forest F

E.3 Algorithm for Edge Analysis

$\Pi_1(e)$	The tail of a data forest edge e
$\Pi_2(e)$	The head of a data forest edge e
$\mathbb{PI}$	The type of a processor input
$\mathbb{PO}$	The type of a processor output
$\mathbb{OI}$	The type of an operation input
$\mathbb{OO}$	The type of an operation output
$\mathbb{WI}$	The type of a (top-level) workflow input
$\mathbb{WO}$	The type of a (top-level) workflow output
$\mathbb{EI}$	The type of an exogenous operator input
V_c	The type of a vertex parameter that consumes data during workflow execution
$v \in V_c$	A consumer parameter v
V_p	The type of a vertex parameter that produces data during workflow execution
$v \in V_c$	A producer parameter v
ς	A binary relation between some consumer parameters and some producer parameters
$dom(\varsigma)$	The domain of the relation ς
ρ	A binary relation between some producer parameters and some consumer parameters
$dom(\rho)$	The domain of the relation ρ
k	The sum of the number of edges in all the data forests involved in a multi-level composition
X_p	A set of producer parameters
Y_c	A set of consumer parameters

E.4 Evaluation (Preliminaries)

G_α	A weighted graph of distributed data flows
V_α	The set of parameters in G_α
$v \in (V_\alpha \cap V_p)$	A producer parameter v in G_α
$v \in (V_\alpha \cap V_c)$	A consumer parameter v in G_α
E_α	The set of directed data flows (between parameters) in G_α
$e \in E_\alpha$	A distributed data flow
$\Omega_\alpha(e)$	The function that maps a distributed data flow $e \in E_\alpha$ to a network communication cost
$\Omega_\alpha((oA, iB))$	The network cost of passing the value from the output parameter oA to the input parameter iB, via a distributed data flow
M	The type of a finite walk of data flows in G_α
$M' \in M$	A finite walk of data flows in G_α
$\alpha(M')$	The function that maps a finite walk $M' \in M$ to the total network cost of routing data from a producer parameter $p \in (V_\alpha \cap V_p)$ to a consumer parameter $q \in (V_\alpha \cap V_c)$
$\omega(v)$	The network cost of writing a value produced by $v \in V_p$
$\Gamma(v)$	The network cost of reading a value for $v \in V_c$
G_β	A weighted graph of decentralised data flows
V_β	The set of parameters in G_β
$v \in (V_\beta \cap V_p)$	A producer parameter v in G_β
$v \in (V_\beta \cap V_c)$	A consumer parameter v in G_β
E_β	The set of directed data flows (between parameters) in G_β
$e \in E_\alpha$	A decentralised data flow
$\Omega_\beta(e)$	The function that maps a decentralised data flow $e \in E_\beta$ to a network communication cost
$\Omega_\beta((oA, iB))$	The network cost of passing the value from the output parameter oA to the input parameter iB, via a decentralised data flow
$\circ$	The operator for function composition
$\Pi_1^\beta(e)$	The tail of a decentralised data flow $e \in E_\beta$
$\Pi_2^\beta(e)$	The head of a decentralised data flow $e \in E_\beta$
$\beta(e)$	The total network cost of routing data from a producer parameter $\Pi_1^\beta(e)$ to a consumer parameter $\Pi_2^\beta(e)$ s.t. $e \in E_\beta$

E.5 Evaluation (Experiments)

τ	The number of steps in the *RQ1* experiment
$f(c)$	The number of distributed data flows when there are $c \in (\mathbb{N} \cap [0, \tau])$ new composites composed by the service S_F
$g(c)$	The number of decentralised data flows when there are $c \in (\mathbb{N} \cap [0, \tau])$ new composites composed by the service S_F
$im(\Omega_\alpha)$	The image of the function Ω_α
$im(\omega)$	The image of the function ω
$im(\Gamma)$	The image of the function Γ

DATA AVAILABILITY

The data underlying this article are available in UCI Machine Learning Repository at `https://archive.ics.uci.edu/ml/datasets/Parking+Birmingham#`.

ACKNOWLEDGEMENTS

This work was partially supported by the H2020 I-BiDaaS project [grant agreement No. 780787].

REFERENCES

[1] Arellanes, D. and Lau, K.-K. (2020) Evaluating IoT service composition mechanisms for the scalability of IoT systems. *Future Generation Computer Systems*, **108**, 827–848.

[2] Sarkar, C., Nambi, A. U., Prasad, R. V., Rahim, A., Neisse, R., and Baldini, G. (2015) DIAT: A Scalable Distributed Architecture for IoT. *IEEE Internet of Things Journal*, **2**, 230–239.

[3] Newman, P. (2020). The Internet of Things 2020. https://www.businessinsider.com/internet-of-things-report?r=US&IR=T (accessed December 14, 2021).

[4] Arellanes, D. (2021) Self-Organizing Software Models for the Internet of Things: Complex Software Structures that Emerge without a Central Controller. *IEEE Systems, Man, and Cybernetics Magazine*, **7**, 4–9.

[5] Botta, A., de Donato, W., Persico, V., and Pescapé, A. (2016) Integration of Cloud computing and Internet of Things: A survey. *Future Generation Computer Systems*, **56**, 684–700.

[6] Giang, N. K., Lea, R., and Leung, V. C. M. (2020) Developing applications in large scale, dynamic fog computing: A case study. *Software: Practice and Experience*, **50**, 519–532.

[7] Paniagua, C., Eliasson, J., and Delsing, J. (2020) Efficient Device-to-Device Service Invocation Using Arrowhead Orchestration. *IEEE Internet of Things Journal*, **7**, 429–439.

[8] Hahn, M., Breitenbücher, U., Kopp, O., and Leymann, F. (2018) Modeling and execution of data-aware choreographies: an overview. *Computer Science - Research and Development*, **33**, 329–340.

[9] Jaradat, W., Dearle, A., and Barker, A. (2016) Towards an autonomous decentralized orchestration system. *Concurrency and Computation: Practice and Experience*, **28**, 3164–3179.

[10] Pantazoglou, M., Pogkas, I., and Tsalgatidou, A. (2014) Decentralized Enactment of BPEL Processes. *IEEE Transactions on Services Computing*, **7**, 184–197.

[11] Barker, A., Weissman, J. B., and Van Hemert, J. I. (2012) Reducing Data Transfer in Service-Oriented Architectures: The Circulate Approach. *IEEE Transactions on Services Computing*, **5**, 437–449.

[12] Sonntag, M., Gorlach, K., Karastoyanova, D., Leymann, F., and Reiter, M. (2010) Process space-based scientific workflow enactment. *International Journal of Business Process Integration and Management*, **5**, 32–44.

[13] Barker, A., Walton, C. D., and Robertson, D. (2009) Choreographing Web Services. *IEEE Transactions on Services Computing*, **2**, 152–166.

[14] Binder, W., Constantinescu, I., and Faltings, B. (2009) Service invocation triggers: a lightweight routing infrastructure for decentralised workflow orchestration. *International Journal of High Performance Computing and Networking*, **6**, 81–90.

[15] Hahn, M., Breitenbücher, U., Leymann, F., Wurster, M., and Yussupov, V. (2018) Modeling Data Transformations in Data-Aware Service Choreographies. *International Enterprise Distributed Object Computing Conference (EDOC)*, Stockholm, Sweden, October, pp. 28–34. IEEE.

[16] Do, T.-X. and Kim, Y. (2017) Control and data plane separation architecture for supporting multicast listeners over distributed mobility management. *ICT Express*, **3**, 90–95.

[17] Mohamed, A., Onireti, O., Imran, M. A., Imran, A., and Tafazolli, R. (2016) Control-Data Separation Architecture for Cellular Radio Access Networks: A Survey and Outlook. *IEEE Communications Surveys & Tutorials*, **18**, 446–465.

[18] Filippini, I., Redondi, A. E. C., and Capone, A. (2017) Beyond Cellular Green Generation: Potential and Challenges of the Network Separation. *Mobile Information Systems*, **2017**, 1–11.

[19] Liu, D. (2002) Data-flow Distribution in FICAS Service Composition Infrastructure. *International Conference on Parallel and Distributed Computing Systems (PDCS)*, Lousville, KY, September, pp. 1–6.

[20] Barker, A., Weissman, J. B., and van Hemert, J. (2008) Eliminating the middleman: peer-to-peer dataflow. *International Symposium on High Performance Distributed Computing (HPDC)*, Boston, MA, USA, June, pp. 55–64. ACM.

[21] Barker, A., Weissman, J. B., and van Hemert, J. (2008) Orchestrating Data-Centric Workflows. *International Symposium on Cluster Computing and the Grid (CCGRID)*, Lyon, France, May, pp. 210–217. IEEE.

[22] Aalst, W. M. P. v. d., Aldred, L., Dumas, M., and Hofstede, A. H. M. t. (2004) Design and Implementation of the YAWL System. In Persson, A. and Stirna, J. (eds.), *Advanced Information Systems Engineering*, Lecture Notes in Computer Science, **3084**. Springer, Berlin, Heidelberg.

[23] OASIS (2007). Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (accessed December 14, 2021).

[24] Morrison, J. P. (1978) Data Stream Linkage Mechanism. *IBM Systems Journal*, **17**, 383–408.

[25] Kahn, G. and Macqueen, D. (1977) Coroutines and Networks of Parallel Processes. *International Federation for Information Processing (IFIP)*, Toronto, Canada, August, pp. 993–998. IFIP.

[26] Cherrier, S., Salhi, I., Ghamri-Doudane, Y. M., Lohier, S., and Valembois, P. (2014) BeC 3: Behaviour Crowd Centric Composition for IoT applications. *Mobile Networks and Applications*, **19**, 18–32.

[27] Seeger, J., Deshmukh, R. A., Sarafov, V., and Bröring, A. (2019) Dynamic IoT Choreographies. *IEEE Pervasive Computing*, **18**, 19–27.

[28] Macker, J. P. and Taylor, I. (2017) Orchestration and analysis of decentralized workflows within heterogeneous networking infrastructures. *Future Generation Computer Systems*, **75**, 388–401.

[29] Giang, N. K., Lea, R., and Leung, V. C. M. (2018) Exogenous Coordination for Building Fog-Based Cyber Physical Social Computing and Networking Systems. *IEEE Access*, **6**, 31740–31749.

[30] Arellanes, D. and Lau, K.-K. (2018) Analysis and Classification of Service Interactions for the Scalability of the Internet of Things. *International Congress on Internet of Things (ICIOT)*, San Francisco, CA, USA, July, pp. 80–87. IEEE.

[31] Autili, M., Inverardi, P., and Tivoli, M. (2018) Choreography Realizability Enforcement through the Automatic Synthesis of Distributed Coordination Delegates. *Science of Computer Programming*, **160**, 3–29.

[32] Wutke, D., Martin, D., and Leymann, F. (2008) Model and infrastructure for decentralized workflow enactment. *ACM Symposium on Applied Computing (SAC)*, Fortaleza, Ceara, Brazil, March, pp. 90–94. ACM.

[33] Giang, N. K., Blackstock, M., Lea, R., and Leung, V. C. M. (2015) Developing IoT applications in the Fog: A Distributed Dataflow approach. *International Conference on the Internet of Things (IOT)*, Seoul, Korea (South), October, pp. 155–162. IEEE.

[34] Arbab, F. (2005) Abstract Behavior Types: a foundation model for components and their composition. *Science of Computer Programming*, **55**, 3–52.

[35] Ghanem, M., Curcin, V., Wendel, P., and Guo, Y. (2008) Building and Using Analytical Workflows in Discovery Net. In Dubitzky, W. (ed.), *Data Mining Techniques on the Grid*. Wiley Publishing, New York.

[36] Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E. A., Tao, J., and Zhao, Y. (2005) Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice & Experience*, **18**, 1039–1065.

[37] Decker, G., Kopp, O., and Barros, A. (2008) An Introduction to Service Choreographies. *Information Technology*, **52**, 122–127.

[38] OpenJS Foundation (2020). Node-RED: Flow-based programming for the Internet of Things.

[39] Ngu, A., Gutierrez, M., Metsis, V., Nepal, S., and Sheng, Q. (2017) IoT Middleware: A Survey on Issues and Enabling Technologies. *IEEE Internet of Things Journal*, **4**, 1–20.

[40] Xue, G., Liu, D., Liu, J., and Yao, S. (2019) A process partitioning technique for constructing decentralized web service compositions. *Software: Practice and Experience*, **49**, 1550–1570.

[41] Cheng, B., Solmaz, G., Cirillo, F., Kovacs, E., Terasawa, K., and Kitazawa, A. (2018) FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities. *IEEE Internet of Things Journal*, **5**, 696–707.

[42] Arellanes, D. and Lau, K.-K. (2019) Workflow Variability for Autonomic IoT Systems. *International Conference on Autonomic Computing (ICAC)*, Umea, Sweden, June, pp. 24–30. IEEE.

[43] Arellanes, D. and Lau, K.-K. (2018) Algebraic Service Composition for User-Centric IoT Applications. In Georgakopoulos, D. and Zhang, L.-J. (eds.), *Internet of Things – ICIOT 2018*, Lecture Notes in Computer Science, **10972**. Springer International Publishing, Cham.

[44] Arellanes, D. and Lau, K.-K. (2017) Exogenous Connectors for Hierarchical Service Composition. *International Conference on Service-Oriented Computing and Applications (SOCA)*, Kanazawa, Japan, November, pp. 125–132. IEEE.

[45] Lau, K.-K. and Di Cola, S. (2017) *An Introduction to Component-based Software Development*, 1st edition. World Scientific, Singapore.

[46] Rana, T., Bangash, Y. A., Baz, A., Rana, T. A., and Imran, M. A. (2020) Incremental Composition Process for the Construction of Component-Based Management Systems. *Sensors*, **20**, 1–18.

[47] Arbab, F., Autili, M., Inverardi, P., and Tivoli, M. (2019) Different Glasses to Look into the Three Cs: Component, Connector, Coordination. In Boreale, M., Corradini, F., Loreti, M., and Pugliese, R. (eds.), *Models, Languages, and Tools for Concurrent and Distributed Programming*. Springer International Publishing, Cham.

[48] Netflix (2020). Conductor. https://netflix.github.io/conductor/ (accessed December 14, 2021).

[49] Arellanes, D. and Lau, K.-K. (2019) Decentralized Data Flows in Algebraic Service Compositions for the Scalability of IoT Systems. *World Forum on Internet of Things (WF-IoT)*, Limerick, Ireland, April, pp. 668–673. IEEE.

[50] Arellanes, D. and Lau, K.-K. (2017) D-XMAN: A Platform For Total Compositionality in Service-Oriented Architectures. *International Symposium on Cloud and Service Computing (SC2)*, Kanazawa, Japan, November, pp. 283–286. IEEE.

[51] Fu, Y. and Soman, C. (2021) Real-time Data Infrastructure at Uber. *International Conference on Management of Data (SIGMOD/PODS)*, China, June, pp. 2503–2516. ACM.

[52] Stolfi, D. H., Alba, E., and Yao, X. (2017) Predicting Car Park Occupancy Rates in Smart Cities. In Alba, E., Chicano, F., and Luque, G. (eds.), *Smart Cities*, Lecture Notes in Computer Science, **10268**. Springer International Publishing, Cham.

[53] Morrison, J. P. (2010) *Flow-Based Programming: A New Approach to Application Development*, 2nd edition. CreateSpace, USA.

[54] Hahn, M., Breitenbucher, U., Leymann, F., and Weiss, A. (2017) TraDE - A Transparent Data Exchange Middleware for Service Choreographies. In Panetto, H., Debruyne, C., Ardagna, C. A., Gaaloul, W., Papazoglou, M., Paschke, A., and Meersman, R. (eds.), *On the Move to Meaningful Internet Systems*, Lecture Notes in Computer Science, **10573**. Springer International Publishing.

[55] Guimaraes, F. P., Kuroda, E. H., and Batista, D. M. (2012) Performance Evaluation of Choreographies and Orchestrations with a New Simulator for Service Compositions. *International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, Barcelona, Spain, September, pp. 140–144. IEEE.

[56] IoT Analytics (2018). State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating. https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/ (accessed December 14, 2021).

[57] Nanda, M. G., Chandra, S., and Sarkar, V. (2004) Decentralizing Execution of Composite Web Services. *ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)*, Vancouver, BC, Canada, October, pp. 170–187. ACM.

[58] Kleinfeld, R., Steglich, S., Radziwonowicz, L., and Doukas, C. (2014) Glue.Things: A Mashup Platform for Wiring the Internet of Things with the Internet of Services. *International Workshop on Web of Things (WoT)*, Cambridge, MA, USA, October, pp. 16–21. ACM.

[59] Chafle, G., Chandra, S., Mann, V., and Nanda, M. G. (2004) Decentralized orchestration of composite web services. *International World Wide Web conference (WWW)*, New York, NY, USA, May, pp. 134–143. ACM.

[60] Zhou, J., Leppanen, T., Harjula, E., Ylianttila, M., Ojala, T., Yu, C., Jin, H., and Yang, L. T. (2013) CloudThings: A common architecture for integrating the Internet of Things with Cloud Computing. *Proceedings of the 2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, Whistler, BC, Canada, June, pp. 651–657. IEEE.