Decentralized Data Flows in Algebraic Service Compositions for the Scalability of IoT Systems

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

***Abstract*—With the advent of the Internet of Things (IoT), scalability becomes a significant concern due to the huge amounts of data generated in IoT systems. A centralized data exchange is not desirable as it leads to a single performance bottleneck. Although a distributed data exchange removes the central bottleneck, it has network performance issues as data passes among multiple coordinators. A decentralized approach is the only solution that fully enables the realization of efficient IoT systems, since there is no single performance bottleneck and network overhead is minimized. In this paper, we present an approach that leverages the semantics of DX-MAN for realizing decentralized data flows in IoT systems. The algebraic semantics of such a model allows a well-defined structure of data flows which is easily analyzed by an algorithm that forms a direct relationship between data consumers and data producers. For the analysis, the algorithm takes advantage of the fact that DX-MAN separates control flow and data flow. Thus, our approach prevents passing data alongside control among multiple coordinators, so data is only read and written on a decentralized data space. We validate our approach using smart contracts on the Blockchain, and conducted experiments to quantitatively evaluate scalability. The results show that our approach scales well with the size of IoT systems.**

***Index Terms*—Internet of Things, decentralized data flows, Blockchain, DX-MAN, exogenous connectors, scalability, separation between control and data, algebraic service composition**

I. Introduction

The Internet of Things (IoT) envisions a world where everything will be interconnected through distributed services. As new challenges are forthcoming, this paradigm requires a shift in our way of building software systems. With the rapid advancement in hardware, the number of connected *things* is increasing considerably, to the extent that scalability becomes a significant concern due to the huge amounts of data involved in IoT systems. Thus, IoT services must exchange data over the Internet efficiently.

Although a centralized data exchange approach has been successful in enterprise systems, it will easily cause a bottleneck in IoT systems which potentially generate a huge amount of data continuously. To avoid the bottleneck, a distributed approach can be used to distribute the load of data over multiple coordinators. However, this would introduce unnecessary network overhead as data is passed among many coordinators.

A decentralized data exchange approach is the most efficient solution to tackle the imminent scale of IoT systems, as it achieves better response time and throughput by minimizing network hops [1], [2], [3], [4], [5], [6], [7]. However, exchanging data among loosely-coupled IoT services is challenging, especially in resource-constrained environments where *things* have poor network connection and low disk space.

Moreover, constructing data dependency graphs is not trivial when control flow and data flow are tightly coupled. The separation between control and data is necessary because it allows a separate reasoning, monitoring, maintenance and evolution of these concerns [8]. Consequently, an efficient data exchange approach can be done without considering control flow, thereby reducing the messages sent over the Internet.

This paper proposes an approach that leverages the semantics of DX-MAN [9], [10] for the realization of decentralized data flows in IoT systems. The algebraic semantics of such a model allows a well-defined structure of data flows which is easily analyzed by an algorithm that forms a direct relationship between data consumers and data producers. For the analysis, the algorithm particularly takes advantage of the fact that DX-MAN separates control flow and data flow.

The rest of the paper is organized as follows. Sect. II introduces the semantics of the DX-MAN model. Sect. III describes DX-MAN data flows and Sect. IV presents the algorithm that analyzes such data flows. Sect. V presents the implementation of our approach. Sect. VI outlines a quantitative evaluation of our approach. Finally, we present the related work in Sect. VII and conclusions in Sect. VIII.

II. DX-MAN Model

DX-MAN is an algebraic model for IoT systems where services and exogenous connectors are first-class entities. An *exogenous connector* is a variability operator that defines multiple workflows with explicit control flow, while a DX-MAN *service* is a distributed software unit that exposes a set of operations through a well-defined interface.

An *atomic service* provides a set of operations and it is formed by connecting an *invocation connector* with a *computation unit*. A computation unit represents an actual service implementation (e.g., a RESTful Microservice or a WS-* service) and it is not allowed to call other computation units. As a consequence of the algebraic semantics, the interface of an atomic service has all the operations in the computation unit, as shown by the red arrows in Fig. 1(a). An invocation

connector defines the most primitive workflow which is the invocation of one operation in the computation unit.

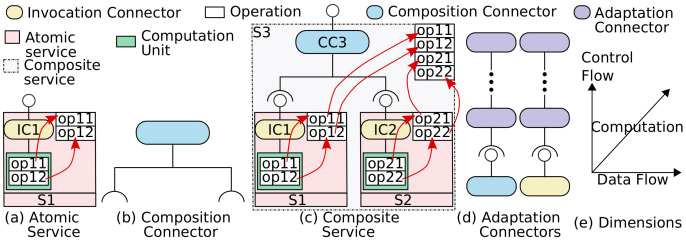

Fig. 1. DX-MAN Model.

Our notion of algebraic service composition is inspired by algebra where functions are hierarchically composed into a new function of the same type. The resulting function can be further composed with other functions so as to yield a more complex one. *Algebraic service composition* is the operation by which a composition connector is used as an operator to compose multiple services, resulting in a (hierarchical) *composite service* whose interface has all the sub-service operations. Thus, a top-level composite will always contain the operations of all the atomic services (Fig. 1(c)). In particular, there are composition connectors for sequencing, branching and parallelism. A *sequencer connector* enables ∞ workflows for the sequential invocation of sub-service operations. A *selector connector* defines $2^n - 1$ branching workflows and chooses the sub-service operations to invoke, such that n is the number of operations in the composite service interface. A *parallel connector* defines ∞ parallel workflows and executes sub-service operations in parallel according to user-defined tasks.

Fig. 1(d) shows that an adapter can be connected with only one exogenous connector. A looping adapter iterates over a sub-workflow while a condition holds true, and a guard adapter invokes a sub-workflow whenever a condition holds true. There are also adapters for sequencing, branching and parallelism which act over the operations of an individual atomic service.

Fig. 1(e) shows that data, control and computation are orthogonal dimensions in DX-MAN. Exogenous connectors enable the separation between control flow and computation, since they decouple service implementations from the (hierarchical) composition structure. Unlike existing composition approaches, data flow never follows control flow as exogenous connectors only pass control to coordinate workflow executions. For further details about the control flow dimension, we refer the reader to our previous papers [9], [10].

III. Data Connectors

A DX-MAN operation is a set of input parameters and output parameters. An input parameter defines the required data to perform a computation, while an output parameter is the data resulting from a specific computation. Although exogenous connectors do not provide any operation (because they do not perform any computation), some of them require data. In particular, selector connectors, selector adapters, looping adapters and guard adapters require input values to evaluate boolean conditions. Exogenous connectors do not have any parameters by default, but designers manually define parameters for a chosen workflow. Workflow selection is out of the scope of this paper, but we refer the reader to our previous paper on workflow variability [10].

In addition to the operations created on algebraic composition, custom operations can be defined in composite services. This is particularly useful when designers want to create a unified composite service interface, in order to hide operations created during algebraic composition.

A *data connector* defines an explicit data flow by connecting a source parameter with a destination parameter. Fig. 2 shows that an *algebraic data connector* is automatically created during composition and is available for all the workflows defined by a composite service. In particular, an algebraic data connector connects two parameters vertically, i.e., bottom-up for outputs and top-down for inputs. Fig. 3 shows the data connection rules for our approach, where we can see that algebraic data connectors can be defined in four different ways only.

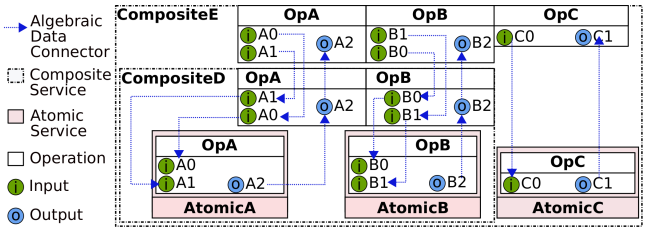

Fig. 2. Algebraic data channels.

Fig. 2 shows that composite services encapsulate data flows to ensure reusability. So, composite services are (black boxes) unaware of data flows of other composites. Hence, there are no data connections between parameters in different composite services, but only data connectors within a composite.

A *custom data connector* is manually created for only one workflow, which connects two parameters either vertically or horizontally. Fig. 3 shows that designers can define custom data connectors in 16 different ways.

Currently, DX-MAN supports custom data connectors for the most common data patterns, namely sequencing and map-reduce. For the sequencing pattern, the parameters of two different operations are horizontally connected. Fig. 4 shows an example of this pattern, where operation *OpB* requires data from operation *OpA*. In particular, a custom data connector links the output *A0* with the input *B0*, while another custom data connector connects the output *A1* with the input *B1*. To improve readability, we omit algebraic data connectors.

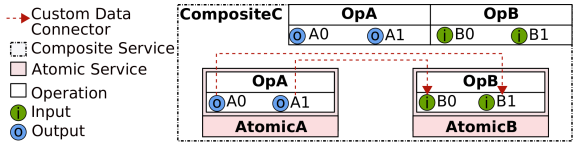

Fig. 4. An example of two sequential data flows.

A *data processor* is particularly useful when data pre-processing needs to be done before executing an operation. It waits until all input values have been received, then performs some computation and returns transformed data in the form of outputs. A *mapper* executes a user-defined function on each

	from\to	Operation in sub-atomic		Operation in sub-composite		Composition connector		Adapter		Operation in the composite		Data processor	
		in	*out*	*in*	*out*	*in*	*out*	*in*	*out*	*in*	*out*	*in*	*out*
Operation in sub-atomic	*in*	✗	✗	✗	✗	✗	*N/A*	✗	*N/A*	✗	✗	✗	✗
	out	**Cust.**	✗	**Cust.**	✗	✗	*N/A*	**Cust.**	*N/A*	✗	**Alge.**	**Cust.**	✗
Operation in sub-composite	*in*	✗	✗	✗	✗	✗	*N/A*	✗	*N/A*	✗	✗	✗	✗
	out	**Cust.**	✗	**Cust.**	✗	✗	*N/A*	**Cust.**	*N/A*	✗	**Alge.**	**Cust.**	✗
Composition connector	*in*	✗	✗	✗	✗	✗	*N/A*	✗	*N/A*	✗	✗	✗	✗
	out	*N/A*											
Adapter	*in*	✗	✗	✗	✗	✗	*N/A*	✗	*N/A*	✗	✗	✗	✗
	out	*N/A*											
Operation in the composite	*in*	**Alge.**	✗	**Alge.**	✗	**Cust.**	*N/A*	**Cust.**	*N/A*	✗	✗	**Cust.**	✗
	out	✗	✗	✗	✗	✗	✗	✗	*N/A*	✗	✗	✗	✗
Data processor	*in*	✗	✗	✗	✗	✗	*N/A*	✗	*N/A*	✗	✗	✗	✗
	out	**Cust.**	✗	**Cust.**	✗	✗	*N/A*	**Cust.**	*N/A*	✗	**Cust.**	**Cust.**	✗

Alge. Algebraic Data Connection

Cust. Custom Data Connection

✗ No Data Connection

N/A No Applicable

Fig. 3. Data connection rules.

input value received and, similarly, a *reducer* takes the result from a mapper and executes a user-defined reduce function on inputs. A *reducer* can also be used in isolation to perform straightforward computation such as combining data into a list. Fig. 5 shows an example of the map-reduce pattern, where operation *opB* requires the pre-processing of data generated by operation *opA*. In particular, two custom data connectors link the input *A0* and the output *A1* with the inputs of the mapper. The output of the mapper is connected to the input of the reducer and, similarly, the output of the reducer is connected to the input *B0*. Note that *A0* can only be connected from the composite service operation, according to Fig. 3.

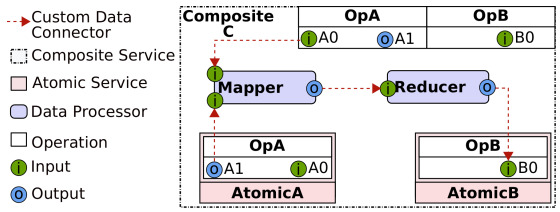

Fig. 5. An example of the map-reduce pattern.

In some workflows, algebraic data connectors may not be needed. For that reason, such connectors can be removed manually at the discretion of the designer. For example, in Fig. 5 all algebraic connectors were removed because data is only needed for the realization of the map-reduce pattern.

IV. Analysis of Data Connectors

Algebraic service composition and the separation of concerns are key enablers of decentralized data flows. In particular, exogenous connectors provide a hierarchical control flow structure that is completely separated from the data flow structure enabled by data connectors. Thus, the data connections in a composite service form a well-structured data dependency graph that is easily analyzed at deployment-time by means of Algorithm 1. To understand this algorithm, some formal definitions are necessary.

Let $\mathbb{D}$ be the data type, $\mathbb{PD}$ the processor parameter type, $\mathbb{OD}$ the operation parameter type and $\mathbb{CD}$ the type of exogenous connector inputs, such that $\mathbb{PD}, \mathbb{OD}, \mathbb{CD} \subseteq \mathbb{D}$. A data connector is then a tuple of type $\mathbb{DC} : \mathbb{D} \times \mathbb{D}$ that connects a $source \in \mathbb{D}$ parameter with a $destination \in \mathbb{D}$ parameter.

Reader parameters are the entities that directly consume data produced by *writer parameters*. I_r is the set of inputs that read data during a workflow execution, namely the inputs of atomic service operations, the inputs of exogenous connectors and the inputs of data processors. O_r is the set of operation outputs in the top-level composite, useful for reading data resulting from a workflow execution. The set I_w represents the required data for a workflow execution, which are the inputs of operations in the top-level composite. O_w is the set of outputs that write data during a workflow execution, namely the outputs of atomic service operations and the outputs of data processors.

Algorithm 1 Algorithm for the analysis of data connectors

1: **procedure** ANALYZE(dc, R, W) ▷ The types of R and W are defined in the text below, and $dc \in \mathbb{DC}$
2: $X_w \leftarrow \emptyset$ ▷ $X_w = \{x \mid x \in \mathbb{D}\}$
3: $Y_r \leftarrow \emptyset$ ▷ $Y_r = \{y \mid y \in \mathbb{D}\}$
4: **if** $\Pi_1(dc) \notin \mathbb{PD} \wedge \Pi_1(dc) \in dom(R)$ **then**
5: $X_w \leftarrow R(\Pi_1(dc))$
6: **else**
7: $X_w \leftarrow \{\Pi_1(dc)\})$
8: **if** $\Pi_2(dc) \notin \mathbb{PD} \wedge \Pi_2(dc) \in dom(W)$ **then**
9: $Y_r \leftarrow W(\Pi_2(dc))$
10: **for each** $y \in Y_r$ **do**
11: $R \oplus \{y \mapsto R(y) - \{\Pi_2(dc)\} \cup X_w\}$
12: **else**
13: $Y_r \leftarrow \{\Pi_2(dc)\})$
14: $R \oplus \{\Pi_2(dc) \mapsto R(\Pi_2(dc)) \cup X_w\}$
15: **for each** $x \in X_w$ **do**
16: $W \oplus \{x \mapsto W(x) \cup Y_r\}$

Basically, Algorithm 1 analyzes data connectors for all composite services, using a bottom-up approach. The goal of this algorithm is to create a relationship between reader parameters and writer parameters, while ignoring those parameters that do not need to manipulate data. To do so, Algorithm 1 receives a data connector $dc \in \mathbb{DC}$ as an input, and uses $R \in ((I_r \cup O_r) \mapsto \{w \mid w \subset I_w \cup O_w\})$ for mapping a reader parameter to a set of writer parameters and $W \in ((I_w \cup O_w) \mapsto \{r \mid r \subset I_r \cup O_r\})$ for mapping a writer parameter to a set of reader parameters.

Algorithm 1 creates two empty sets X_w and Y_r for analyzing the endpoints of a data connector dc. X_w is the set of parameters connected to the *source parameter* $\Pi_1(dc)$ iff $\Pi_1(dc)$ is not a data processor parameter and has incoming data connectors; otherwise, X_w only contains $\Pi_1(dc)$. Similarly, if the *destination parameter* $\Pi_2(dc)$ is not a data processor parameter and $\Pi_2(dc)$ has outgoing data connectors, then Y_r is

the set of parameters connected from $\Pi_2(dc)$ and X_w (without $\Pi_2(dc)$) is added into the writers of each element $y \in Y_r$; otherwise, Y_r only contains $\Pi_2(dc)$ and X_w is added into the writers of $\Pi_2(dc)$. Finally, the set Y_r is added into the readers of each element $x \in X_w$. The result of the algorithm is thus a mapping of reader parameters to writer parameters.

V. Implementation

We implemented our approach on top of the DX-MAN Platform [11], and we used the Blockchain as the underlying data space for persisting parameter values, and for leveraging the capabilities provided by these decentralized platforms, such as performance, security and auditability. Furthermore, using the Blockchain ensures that every service is the owner of its own data, while data provenance is provided to discover data flows (i.e., how data is moved between services) or how parameters change over time. In particular, we defined three smart contracts using *Hyperledger Composer 0.20.0* for executing transactions on *Hyperledger Fabric 1.2*. We do not show the source code due to space constraints, but it is available at https://gitlab.cs.man.ac.uk/mbaxrda2/dxman/tree/development.

The DX-MAN platform provides an API to support the three phases of a DX-MAN system lifecycle: design-time, deployment-time and run-time. Composite service templates only contain algebraic data connectors, as they represent a general design with multiple workflows. Using API constructs, a designer chooses a workflow and defines custom data connectors (and perhaps data processors) for each composite service involved. Similarly, data processor functions are defined by designers using API constructs.

At deployment-time, Algorithm 1 analyzes data connectors (defined at design-time), in order to construct a Java HashMap for readers where the keys are reader parameter UUIDs and the values are lists of writer parameter UUIDs. After getting the map for a given workflow, reader parameters (with their respective list of writers) are stored as assets in the Blockchain by means of the transaction *CreateParameters*.

At run-time, exogenous connectors coordinate a workflow execution by passing control via CoAP messages. When control reaches an invocation connector, the five steps illustrated in Fig. 6 are performed. Although the rest of exogenous connectors behave similarly, they only perform the first two steps. First, the invocation connector uses the transaction *readParameters* to read input values from the Blockchain. For each input, the Blockchain reads values directly from the writers list. As there might be multiple writer parameters, this transaction returns a list of the most recent values that were updated during the workflow execution. Hence, a timestamp is set whenever a parameter value is updated. Output values are written onto the data space as soon as they are available, even before control reaches data consumers. Thus, having concurrent connectors (e.g., a parallel connector) may lead to synchronization issues during workflow execution. To solve this, control flow blocks in the invocation connector until all input values are read.

Once all inputs are ready, the invocation connector executes the implementation of an operation by passing the respective

Fig. 6. Steps for the invocation of an operation implementation.

input values. Then, the operation performs some computation and returns a result in the form of outputs. Finally, the invocation connector writes the output values onto the Blockchain using the transaction *updateParameters*.

An *UpdateParameterEvent* is published whenever a parameter value is updated. At deployment-time, the DX-MAN platform subscribes data processor instances to the events produced by the respective writer parameters. Thus, a data processor instance waits until it receives all events, before performing its respective designer-defined computation. Although our current implementation supports only *mappers* and *reducers*, further data processors can be introduced using the semantics presented in Sect. III, e.g., a *shuffler* can be added to sort data by key.

Our approach enables transparent data exchange as data routing is embodied in the Blockchain. Thus, reader parameters are not aware where the data comes from, and writer parameters do not know who reads the data they produce. Furthermore, the map generated by Algorithm 1 avoids the inefficient approach of passing values through data connectors during workflow execution. Thus, exogenous connectors and data processors read data directly from parameters which only write values onto the Blockchain. Undoubtedly, this enables a transparent decentralized data exchange.

VI. Evaluation

In this section, we present a comparative evaluation between distributed data flows and decentralized data flows for a DX-MAN composition. In the former approach, data is passed over the network through data connectors, whereas the second approach is our proposal. Our evaluation intends to answer two major research questions: (A) Does the approach scale with the number of data connectors? and (B) Under which conditions is decentralized data exchange beneficial?

As a DX-MAN composition has a multi-level hierarchical structure, an algebraic data connector passes a data value vertically in a bottom-up way (for inputs) or in a top-down fashion (for outputs) while a custom data connector passes values horizontally or vertically. For our evaluation, we only consider vertical routing through algebraic data connectors.

$M_p = \{\lambda_j | \lambda_j \in \mathbb{R}\}$ is the set of network message costs for vertically routing the value of a parameter p, where λ_j is the cost of passing a value for p through an algebraic data connector j. Likewise, Γ_p and ω_p are the costs of reading and writing a value for p on the data space, respectively.

Equations 1 and 2 calculate the total message cost of routing a value p using a distributed approach s.t. α_p is for input values and β_p is for output values. Remarkably, as the decentralized approach does not pass values through data connectors, its total message cost of routing the value of p is Γ_p for inputs, and ω_p for outputs.

$$\alpha_p(\Gamma_p, M_p) = \Gamma_p + \sum_{j=0}^{|M_p|-1} (\lambda_j \in M_p) \quad (1)$$

$$\beta_p(\omega_p, M_p) = \omega_p + \sum_{j=0}^{|M_p|-1} (\lambda_j \in M_p) \quad (2)$$

Fig. 7 depicts the DX-MAN composition that we consider for our evaluation, which has three levels, three atomic services and two composite services. The composites *ServiceD* and *ServiceE* have three and five data connectors, respectively. Fig. 7 shows that a data connector has a cost $\lambda_{j\in[0,7]}$ of passing a value over the network. Then, the vertical routing sets for the parameters are $M_{A0} = \{\lambda_3\}$, $M_{A1} = \{\lambda_4\}$, $M_{B0} = \{\lambda_0, \lambda_5\}$, $M_{B1} = \{\lambda_1, \lambda_6\}$ and $M_{C0} = \{\lambda_2, \lambda_7\}$.

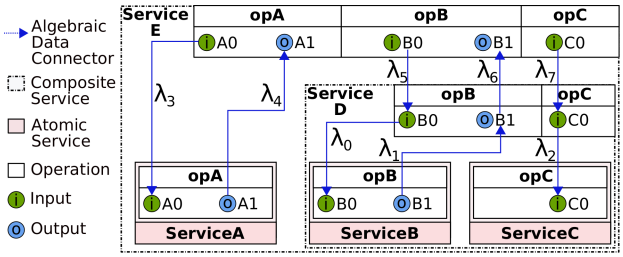

Fig. 7. DX-MAN composition for the evaluation of our approach.

For clarity, we assume that the DX-MAN composition interacts with an external application via a shared data space. So, we can ignore the cost of passing data between the application and the composition. The costs of reading the inputs *A0*, *B0* and *C0* are Γ_{A0}, Γ_{B0} and Γ_{C0}, respectively, and the costs of writing the outputs *A1* and *B1* are ω_{A1} and ω_{B1}, respectively.

Suppose that a specific workflow requires the invocation of the operations *opA* and *opC*. Using a distributed exchange would require passing and reading values for two inputs, and returning and writing one output value. Therefore, the total message cost would be $\alpha_{A0} + \beta_{A1} + \alpha_{C0} = \lambda_3 + \lambda_4 + \lambda_2 + \lambda_7 + \Gamma_{A0} + \omega_{A1} + \Gamma_{C0}$. Remarkably, the total message cost using the decentralized exchange would be $\Gamma_{A0} + \omega_{A1} + \Gamma_{C0}$.

A. RQ1: Does the approach scale with the number of data connectors?

We conducted an experiment that dynamically increases the number of data connectors of the DX-MAN composition depicted in Fig. 7. The experiment is carried out in 100000 steps with $\Gamma_{A0} = \omega_{A1} = \Gamma_{B0} = \omega_{B1} = \Gamma_{C0} = 1$.

For each step of the experiment, we add a new parameter in a random atomic operation. As a consequence of algebraic composition, another parameter is added in the respective composite operation and a data connector links these parameters.

In this experiment, we compare the cost of the distributed exchange vs. the cost of the decentralized exchange. Rather than computing the costs for the invocation of specific operations, we compute the total costs for the DX-MAN composition using $\Gamma_{A0} + \omega_{A1} + \Gamma_{B0} + \omega_{B1} + \Gamma_{C0} + \sum_{j=0}^{7} \lambda_j$. Fig. 8 shows that the costs grow linearly with the number of data connectors, and that the decentralized approach outperforms its counterpart by reducing costs by a factor of 2.67 in average.

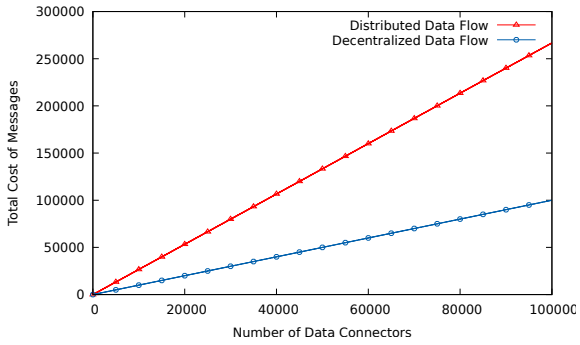

Fig. 8. Impact of increasing the number of data connectors in a DX-MAN composition.

B. RQ2: Under which conditions is decentralized data exchange beneficial?

We conducted an experiment of 100000 steps to see the benefit of the decentralized approach as the number of levels of the composition increases. We particularly consider the total costs for the input *A0* and we assume that $\Gamma_{A0} = 1$. At each step, the number of levels is increased by 1 and $\sum_{j=0}^{|M_{A0}|-1} \lambda_j$ by 0.0004. Thus, increasing the sum of vertical costs means that $\frac{\sum_{j=0}^{|M_{A0}|-1} \lambda_j}{|M_{A0}|} = 1$ and increasing the number of levels by 1 means that $|M_{A0}|$ is also increased by 1. The improvement rate of the decentralized data exchange is $1 - \frac{\Gamma_{A0}}{(\Gamma_{A0} + \sum_{j=0}^{|M_{A0}|-1} \lambda_j)}$.

Fig. 9 shows the results of this experiment, where it is clear that the benefit of the decentralized approach becomes more evident as the number of levels of the composition increases. This is because the number of data connectors increases with the number of levels and so the cost of the distributed approach. The only way a distributed approach would outperform the decentralized one is when the cost of performing operations on the data space is more expensive than the total cost of passing values vertically. In particular, for our experiment the DX-MAN composition would get a benefit only if $\Gamma_{A0} < \sum_{j=0}^{|M_{A0}|-1} \lambda_j$.

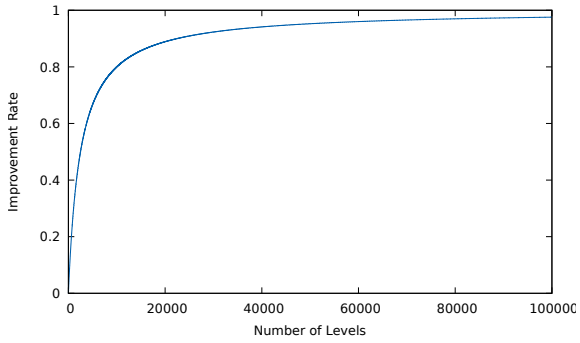

Fig. 9. Impact of increasing the number of levels in a DX-MAN composition.

VII. Related Work

This section presents the related work on decentralized data flows in service-oriented systems. We classified our findings into three categories, depending on the composition semantics the approaches are built on: orchestration (with central control flows and decentralized data flows), decentralized orchestration, data flows and choreographies.

Orchestration approaches [12], [1] partially separate data from control so as to enable P2P data exchanges. To do so, an orchestrator coordinates the exchanges by passing data references alongside control. Thus, extra network traffic is introduced as data references (and acknowledge messages) are transferred over the network. These approaches are typically based on proxies that keep data, thus causing an issue for *things* with low disk space. By contrast, DX-MAN does not require any coordinator for data exchange, and exogenous connectors do not store data. Besides, exogenous connectors do not exchange references, thanks to the separation of concerns.

Only few approaches discuss data decentralization using the semantics of decentralized orchestration [5], [7]. [13] stores data and control in distributed tuple spaces which may become a bottleneck in IoT environments continuously generating huge amounts of data. [3] stores references instead of values, but references are still needed because data is mixed with control. Moreover, [3] requires the maintenance of tuple spaces (for passing references), and databases (for storing data). In DX-MAN, references and values coexist in the same data space.

Although exogenous data flows [14] allocate flows over different *things*, there is a master engine that coordinates data exchange for slave engines. Hence, this approach introduces extra network hops as data is passed among multiple engines. Service Invocation Triggers [2] use endogenous data flows to exchange data directly, but they rely on workflows that do not contain loops and conditionals. This limitation arises from the fact that it is not trivial to analyze data dependencies when control is mixed with data. In general, endogenous data flows [15], [16] do not support explicit control flow which is a crucial requirement for the scalability of IoT systems [8].

A choreography describes interactions among participants using decentralized message exchanges (a.k.a. conversations). Workflow participants [17] pass data among multiple engines leading to network degradation. Although services may exchange data directly by message passing, they are not reusable because data and control are mixed [8]. [4] uses peers to separate control from computation; however, peers pass data alongside control according to predefined conversations, leading to the issues discussed in [6]. Although [18] proposes the separation of control and data for choreographies, it uses a middleware which may potentially become a central bottleneck.

VIII. Conclusions

In this paper, we presented an approach on top of DX-MAN for decentralized data flows in IoT systems. At design-time, the algebraic semantics of DX-MAN enables a well-defined structure of data connections. As data connections are not mixed with control flow structures, an algorithm easily analyzes data connections at deployment-time. The result is a mapping between reader parameters and writer parameters, which prevents passing values through data connectors. In our current implementation, the Blockchain embodies this mapping to manage data values at run-time.

DX-MAN is a service model that separates data flow, control flow and computation, for a separate reasoning, monitoring, maintenance and evolution of such concerns. Separating data and control particularly prevents exogenous connectors from passing data alongside control, and allows the use of different technologies to handle data flows and control flows separately.

Our experiments confirm that our approach scales well with the number of data connectors and with the number of levels of a DX-MAN composition. They also suggest that our approach provides the best performance when the cost of performing operations on the data space is less than the total cost of passing data over the network. Thus, our approach is extremely beneficial for IoT systems consisting of plenty of services.

Acknowledgment

This research is sponsored by the National Council of Science and Technology (CONACyT).

References

[1] A. Barker *et al.*, "Reducing Data Transfer in Service-Oriented Architectures: The Circulate Approach," *IEEE Trans. Serv. Comput.*, vol. 5, no. 3, pp. 437–449, 2012.

[2] W. Binder *et al.*, "Service invocation triggers: a lightweight routing infrastructure for decentralised workflow orchestration," *Int. J. High Perf. Comp. and Net.*, vol. 6, no. 1, pp. 81–90, 2009.

[3] M. Sonntag *et al.*, "Process space-based scientific workflow enactment," *Int. J. Business Proc. Integr. and Man.*, vol. 5, no. 1, pp. 32–44, 2010.

[4] A. Barker *et al.*, "Choreographing Web Services," *IEEE Trans. Serv. Comput.*, vol. 2, no. 2, pp. 152–166, 2009.

[5] M. Pantazoglou *et al.*, "Decentralized Enactment of BPEL Processes," *IEEE Trans. Serv. Comput.*, vol. 7, no. 2, pp. 184–197, 2014.

[6] M. Hahn *et al.*, "Data-Aware Service Choreographies Through Transparent Data Exchange," in *Web Eng.*, ser. Lect. Notes Comp. Sci. Springer Int. Pub., 2016, pp. 357–364.

[7] W. Jaradat *et al.*, "Towards an autonomous decentralized orchestration system," *Concurr. Comp. Pract E.*, vol. 28, no. 11, pp. 3164–3179, 2016.

[8] D. Arellanes and K.-K. Lau, "Analysis and Classification of Service Interactions for the Scalability of the Internet of Things," in *IEEE ICIOT*, 2018, pp. 80–87.

[9] D. Arellanes and K.-K. Lau, "Exogenous Connectors for Hierarchical Service Composition," in *IEEE SOCA*, 2017, pp. 125–132.

[10] D. Arellanes and K.-K. Lau, "Algebraic Service Composition for User-Centric IoT Applications," in *ICIOT 2018*, ser. Lect. Notes Comp. Sci. Springer Int. Pub., 2018, pp. 56–69.

[11] D. Arellanes and K.-K. Lau, "D-XMAN: A Platform For Total Compositionality in Service-Oriented Architectures," in *IEEE SC2*, 2017, pp. 283–286.

[12] D. Liu, "Data-flow Distribution in FICAS Service Composition Infrastructure," 2002.

[13] D. Wutke *et al.*, "Model and Infrastructure for Decentralized Workflow Enactment," in *Proc. Symp. on Appl. Comp.* ACM, 2008, pp. 90–94.

[14] N. K. Giang *et al.*, "Developing IoT applications in the Fog: A Distributed Dataflow approach," in *IOT*, 2015, pp. 155–162.

[15] J. Seeger *et al.*, "Running Distributed and Dynamic IoT Choreographies," in *IEEE GIoTS*, 2018, pp. 1–6.

[16] S. Cherrier *et al.*, "D-LITe: Distributed Logic for Internet of Things Services," in *IEEE iThings/CPSCom*, 2011, pp. 16–24.

[17] G. Decker *et al.*, "BPEL4chor: Extending BPEL for Modeling Choreographies," in *IEEE ICWS*, 2007, pp. 296–303.

[18] M. Hahn *et al.*, "TraDE - A Transparent Data Exchange Middleware for Service Choreographies," in *On the Move to Meaningful Internet Syst.*, ser. Lect. Notes Comp. Sci. Springer Int. Pub., 2017, pp. 252–270.