Algebraic Service Composition for User-Centric IoT Applications

Damian Arellanes(✉) and Kung-Kiu Lau

School of Computer Science, The University of Manchester,
Manchester M13 9PL, UK
{damian.arellanesmolina,kung-kiu.lau}@manchester.ac.uk

Abstract. The Internet of Things (IoT) requires a shift in our way of building applications, as it is aimed at providing many services to society in general. Non-developer people require increasingly complex IoT applications and support for their ever changing run-time requirements. Although service composition allows the combination of functionality into more complex behaviours, current approaches provide support for dealing with one IoT scenario at a time, as they allow the definition of only one workflow. In this paper, we present DX-MAN, an algebraic model for static service composition that allows the definition of composite services that encompass multiple workflows for run-time scenarios. We evaluate our proposal on an example in the domain of smart homes.

Keywords: IoT applications · Algebraic service composition
Scalability · Exogenous connectors · End-user development · DX-MAN

1 Introduction

The Internet of Things (IoT) promises a new era in which every physical world object and all living entities will be interconnected through innovative distributed services. Thus, the scale of IoT applications will go beyond human mind expectations.

IoT applications are mainly aimed at providing value to society in general. People with no development expertise are able to control, manage and customize their own applications [6,11]. For this reason, IoT requires a shift in our way of building applications: a developer must be able to create a generic application that encompasses multiple scenarios, in order to accommodate as much as possible the run-time user requirements. Thus, users will be able to autonomously choose a behaviour among the alternative ones.

Although some scenarios are simple, many others require the combination of a huge number of services. Hence, service composition is crucial for building complex IoT applications. However, designing a generic composite that accommodates multiple IoT scenarios is not trivial, since user requirements may vary from one scenario to another. Moreover, the dynamism of IoT applications causes an increase in the number of possible scenarios as the number of services grows.

D. Georgakopoulos and L.-J. Zhang (Eds.): ICIOT 2018, LNCS 10972, pp. 56–69, 2018.
https://doi.org/10.1007/978-3-319-94370-1_5

Current composition approaches do not fulfill the demands of IoT applications that require user-centric compositions of a huge number of services. This is because their semantics allows the definition of only one workflow at a time. Thus, tackling a new scenario would require an entirely new workflow or the modification of an existing one. This paper proposes DX-MAN, an algebraic model for static IoT service composition, which enables the development of composite services that encompass many workflows so as to accommodate multiple run-time scenarios.

The rest of the paper is structured as follows. Section 2 presents the related work. Section 3 presents a motivating example. Section 4 describes our model for algebraic IoT service composition. Section 5 presents examples to show the feasibility of our model. Section 6 presents a discussion of our results as well as challenges related to this research. Finally, Sect. 7 presents the conclusions and the future work.

2 Related Work

Current composition approaches include orchestration, nested orchestrations, choreography, data flows and nested data flows.

Orchestration [13,22] and *choreography* [8,26,27] have been used for many years in Service Oriented Architecture (SOA) and are now gaining attention for IoT applications. Orchestration defines a central coordinator for the invocation of operations in services. In order to eliminate the performance bottleneck caused by the central coordinator or to support multiple administrative domains, a number of sub-workflows can be defined in *nested orchestrations* [7,16]. On the other hand, a choreography realizes a workflow through the collaborative and decentralized exchange of messages between the services involved. Regardless of the underlying mechanics, (nested) orchestration and choreography allow the definition of only one workflow at a time.

A data-driven workflow, or *data flow*, allows the combination of data streams from different IoT sources. It is basically a graph where nodes represent computation and edges represent data paths: a node receives data, then performs some computation and finally passes data on. Although data flows are increasingly popular for IoT applications thanks to the emergence of *mashups* [5,14,24], they allow the creation of one workflow at a time; and this is also true in *nested data flows* [12].

Like orchestration, data flows are considered as exogenous composition mechanisms because a workflow is defined with no knowledge of the services involved [18]. Reo [19,23] is a declarative language for data flows, which also has the notion of exogenous connectors. Unlike DX-MAN, the composition of two Reo connectors yields a more complex connector, but not a service. Of course, a Reo connector can be transformed into a service, but this would require an extra step as it is not part of Reo semantics. More importantly, Reo allows the creation of one workflow at a time, like other data flow approaches.

Automatic service composition [1,9,10,22] consists of discovering, selecting and combining services at run-time, in order to construct a workflow that fulfills

a given specification [17,25,28]. It does not provide new composition semantics, but it is built on top of existing ones: data flows [9], orchestration [22], choreography [1], or any combination thereof [10]. Therefore, automatic service composition also allows the definition of only one workflow at a time.

Other approaches [11,15,29] do not provide any composition constructs, because they are only *frameworks* or *software tools* for end-user development. Some of them provide support to define only one straightforward workflow at a time, typically a sequential one.

3 Motivating Example

To motivate our approach, this section introduces a running example. The example is in the domain of smart homes and is based on the case study presented in [22]. It consists of two independent services shown in Fig. 1: (i) a *Windows* service for opening and closing windows and (ii) a *Climate* service to turn dehumidifiers on and off. For simplicity, we only show two operations per service. The distribution of services over IoT nodes is out of the scope of this paper.

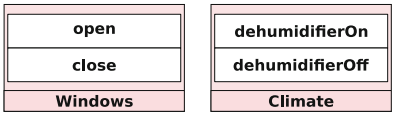

Fig. 1. Services involved in our motivating example.

Imagine a user requires different workflows at run-time depending on climatic conditions. Automatic service composition is the best approach currently available to generate workflows on the fly. However, it allows the definition of only one workflow at a time, since it is built on top of existing composition semantics.

For example, on a sunny day the user may want to open the windows and turn the dehumidifier off. The workflow depicted in Fig. 2 is generated by an automatic composition mechanism so as to accommodate this user requirement. Suppose it suddenly starts raining so the user decides to close the windows and turn the dehumidifier on. Thus, the automatic composition mechanism would need the generation of the entirely new workflow shown in Fig. 3.

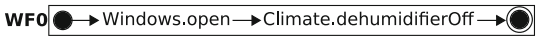

Fig. 2. Workflow for a sunny day.

Of course, the user can express all his needs in a single step. The workflow generated by the automatic composition mechanism for this scenario is shown in

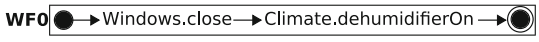

Fig. 3. Workflow for a rainy day.

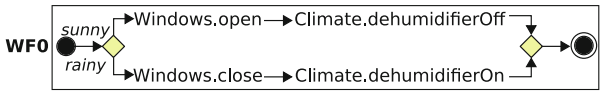

Fig. 4. Workflow for a sunny and rainy day.

Fig. 4, and it includes the scenarios depicted in Figs. 2 and 3. If the user changes his mind again, a new workflow would be needed.

It might seem that nested workflows are an alternative solution to this problem, as shown in Fig. 5. However, their composition semantics also allow the definition of only one workflow at a time. Thus, individual nested workflows are required whenever user requirements change.

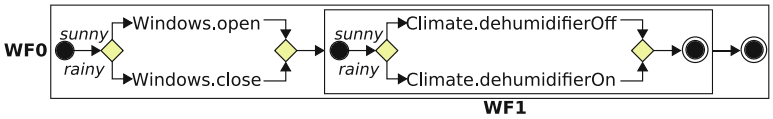

Fig. 5. Nested workflows for a sunny and rainy day.

4 DX-MAN

We propose DX-MAN (Distributed X-MAN) [3] to mitigate the impact of change of run-time user requirements. It is a multi-level service composition model [4] inspired by algebra and the X-MAN component model [20,21], where services and exogenous connectors are first-class entities. Figure 6 illustrates the DX-MAN constructs which we further describe in this section.

A DX-MAN service is a distributed software unit that exposes a set of operations through a well-defined interface. It can be deployed in any IoT node such as a Cloud, an edge device or a sensor. Distribution semantics are out of the scope of this paper, but we refer the reader to another paper on that matter [3].

An atomic service is the most primitive kind of DX-MAN service. It is formed by connecting an invocation connector with a computation unit (see Fig. 6). The invocation connector provides access to the operations implemented in the computation unit, and the computation unit is not allowed to call other computation units. The atomic service interface has all the operations implemented in the computation unit. Formally, an atomic service $AS \in \mathbb{S}$, where $\mathbb{S}$ is the type of services, is a set of operations defined as follows:

$$AS = \{op_i \mid i \in \mathbb{N}\} \tag{1}$$

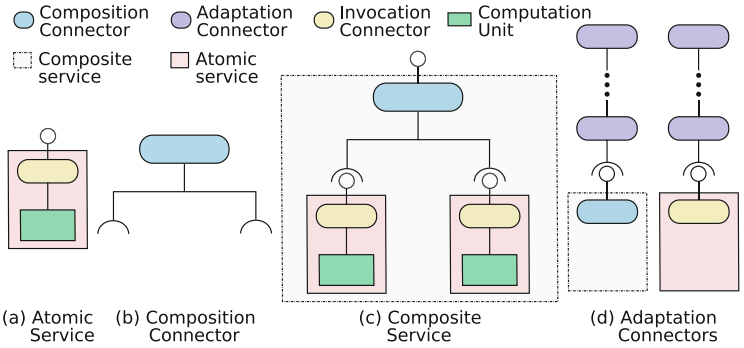

Fig. 6. DX-MAN constructs.

Exogenous connectors are architectural elements that define explicit control flow and encapsulate a network communication mechanism, in order to coordinate the execution of an IoT application from outside services. So, services are unaware they are part of a larger piece of behaviour.

Our notion of algebraic composition is inspired by algebra where functions are composed hierarchically into a new function of the same type, using the operator $\circ$. The resulting function can be further composed with other functions, yielding a more complex one.

Algebraic service composition means that a composition connector is used as an operator ($\circ$) *to hierarchically compose* ≥ 1 *services, atomic or composite, into a (composite) service. As it is constructed from sub-service interfaces, the composite interface has all the sub-service operations. Like an algebraic function, a composite service is a generalization of a particular problem because it implicitly contains multiple workflows whose formation is constrained by the composition connector being used.* Formally, a composite service $CS \in \mathbb{S}$, where $\mathbb{S}$ is the type of services, is a set of services defined as follows:

$$CS = \{S_i \mid i \in \mathbb{N} \wedge S \in \mathbb{S}\} \tag{2}$$

DX-MAN provides composition connectors for sequencing, branching and parallelism. A sequencer connector (*SEQ*) allows the invocation of sub-service operations in a user-defined order. A sub-service operation can be associated with ≥ 0 orders. Sub-service operations with no given order are never invoked, and when no sub-service operation has an order assigned, an empty workflow is thrown at run-time. Any sub-service operation can be invoked any number of times within a workflow. Thus, a sequencer connector defines a composite service that contains an infinite number of sequential workflows. Figure 7 shows an example of a composite service constrained by a sequencer connector.

A selector (*SEL*) connector chooses the sub-service operations to be invoked, according to user-defined conditions which are evaluated concurrently. A sub-service operation can be associated with exactly zero or one condition. Sub-service operations with no condition associated are never invoked. When no

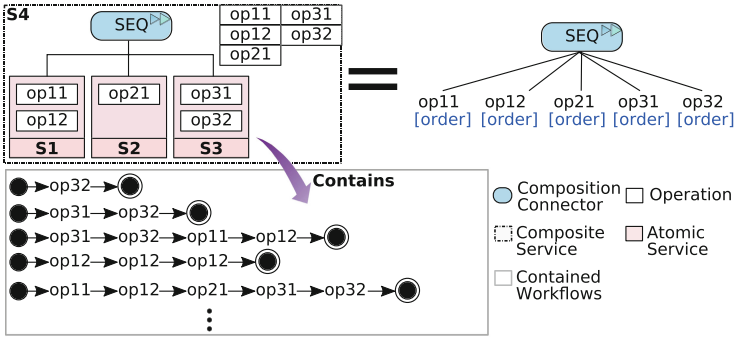

Fig. 7. Sequencer connector.

sub-service operation has a condition associated or all conditions hold false, an empty workflow is thrown at run-time. A selector connector defines a composite service that contains $2^{|\bigcup_{i=1}^{|CS|} S_i|}$ workflows. For example, Fig. 8 shows a composite service that contains 32 possible branching workflows as there are five sub-service operations. We do not show all possible workflows because of space constraints.

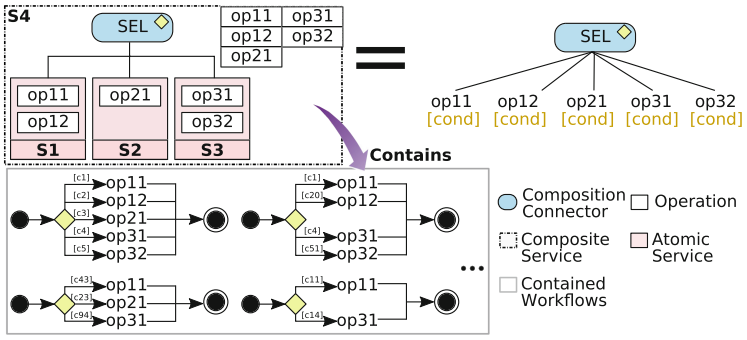

Fig. 8. Selector connector.

A parallel connector (PAR) allows the parallel invocation of sub-service operations. A sub-service operation can be invoked multiple times in parallel within a workflow; to do so, the user needs to specify the number of jobs for each sub-service operation. When no sub-service operation has jobs assigned, an empty workflow is thrown at run-time. A parallel connector defines a composite service that contains infinite parallel workflows. Figure 9 shows a composite service constrained by a parallel connector.

Although they do not compose services, adapters can also constrain workflows by applying additional control structures over an individual service. A looping adapter can be used to iterate a number of times over a sub-workflow, while a

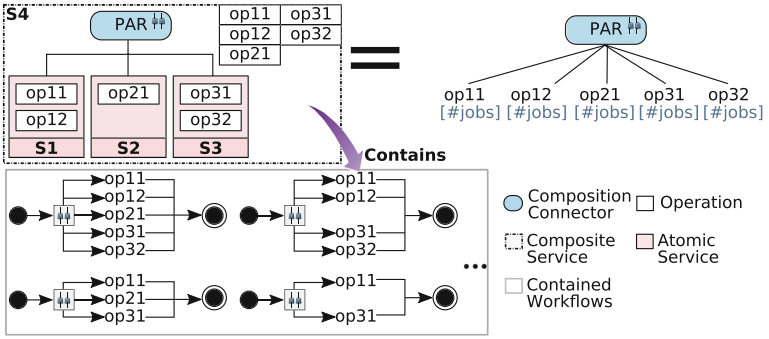

Fig. 9. Parallel connector.

user-defined condition holds true. A guard adapter invokes a sub-workflow only if a user-defined condition is true.

Selection trees are abstract templates that allow the selection of workflows at run-time. They are implicitly created from a composite service during design-time. Figures 7, 8 and 9 show examples of selection trees for a sequencer connector, selector connector and parallel connector, respectively. In the next section, we present examples that show how to choose workflows using selection trees.

5 Examples

This section presents two examples of using DX-MAN for user-centric IoT applications. The first example describes how a one-level composite service accommodates the run-time user requirements described in our motivating example (see Sect. 3). The second example describes how a two-level composite service enables more complex workflows by hierarchically composing services. For both examples, we distinguish between developers and users. Developers design, deploy and execute DX-MAN services, while users choose the workflow they need at run-time. To do so, we developed a platform prototype [2].[1] Composite services and selection tree instances are defined using JavaScript Object Notation (JSON) documents. Due to space constraints and clarity, we omit the JSON documents used for the examples. Instead, we show a graphical representation of composite services and selection trees.

5.1 One-Level Composition

At design-time, the developer uses a sequencer connector *SEQ0* to compose the services *Windows* and *Climate* into a composite service *C0* which contains infinite sequential workflows (see Fig. 10). At run-time, the user only chooses the workflow he needs from the composite *C0*.

[1] https://gitlab.cs.man.ac.uk/mbaxrda2/DX-MAN.

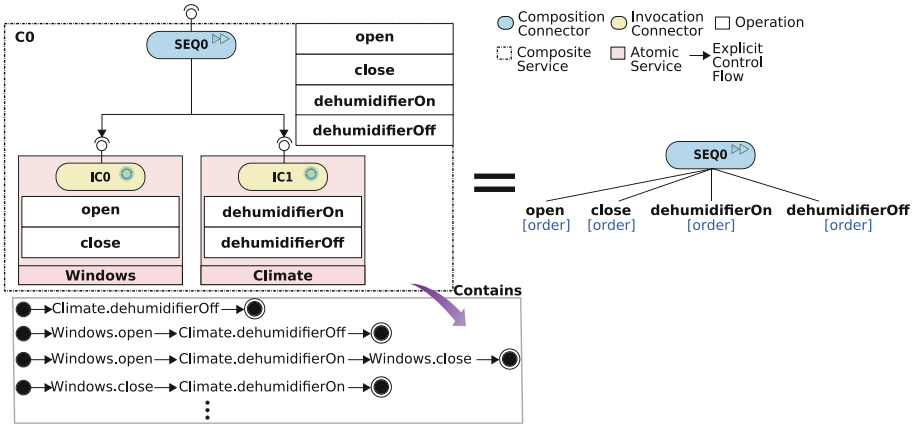

Fig. 10. DX-MAN architecture for the scenarios of our motivating example.

For example, on a sunny day the user chooses the workflow depicted in Fig. 2 in Sect. 3, by assigning execution order as shown in Fig. 11. Suddenly, it starts raining so the user chooses the workflow illustrated in Fig. 3 in Sect. 3, by assigning execution order as shown in Fig. 12. Thus, there is clearly no need of creating an individual workflow or a new composite service whenever user requirements change, but only defining an instance of the respective selection tree.

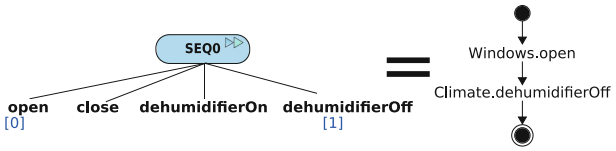

Fig. 11. Choosing a workflow for a sunny day.

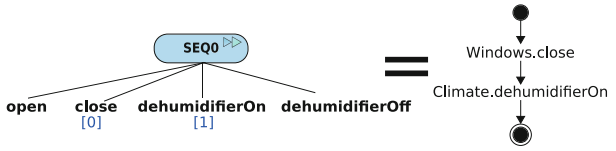

Fig. 12. Choosing a workflow for a rainy day.

As another example, on a cold day the user may want to only close the windows. To do so, the user assigns the execution order shown in Fig. 13. Again, without the need of creating an individual workflow or a new composite service.

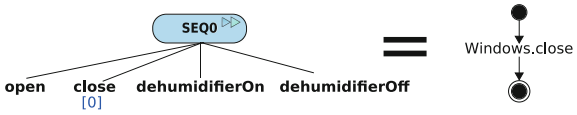

Fig. 13. Choosing a workflow for a cold day.

5.2 Two-Level Composition

In the previous subsection, we presented a one-level composition as a solution for our motivating example. Nevertheless, DX-MAN allows more complex workflows by hierarchically composing services into multi-level structures.

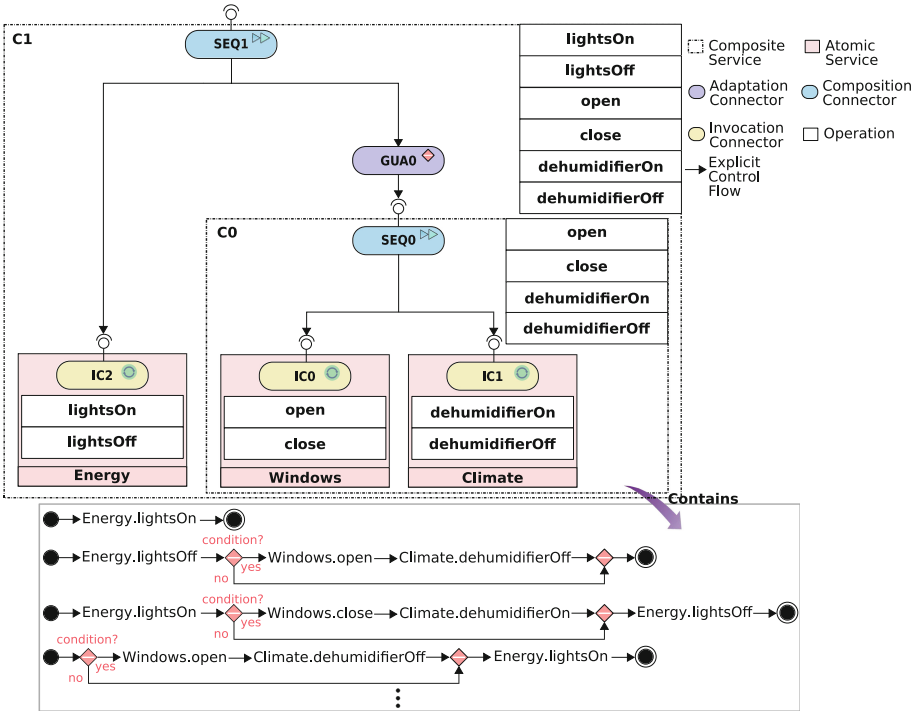

Fig. 14. Two-level DX-MAN architecture.

Suppose there is an atomic service *energy* for turning lights on and off. The developer uses a sequencer *SEQ1* to compose the existing composite *C0* and the atomic service *energy* into a new composite *C1*. He also adds a guard adapter to invoke *C0* if a user-defined condition holds true. Figure 14 shows the resulting two-level DX-MAN composition, and Fig. 15 shows the respective selection tree.

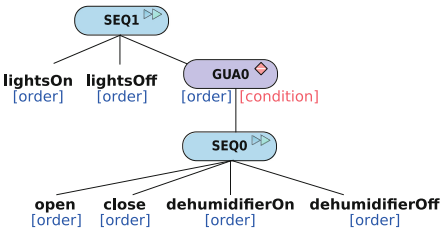

Fig. 15. Resulting tree from the two-level DX-MAN architecture.

Unlike nested workflows, a DX-MAN composite service enables an entirely new world of alternative workflows as shown in Fig. 14. For example, the user may want the following workflow before sleeping: turn the lights off and, if it all the lights were successfully turned off, close the windows and turn the dehumidifier off. To choose that workflow from *C1*, the user assigns the execution order shown in Fig. 16. A condition is represented as a JSON document and specifies the name of the parameter, the operator (only "==" and "!=" are supported at this stage) and the value to compare with. For example, the condition for *GUA0* would be {*"parameterName":"lightsStatus","operator":"==","value":"off"*}.

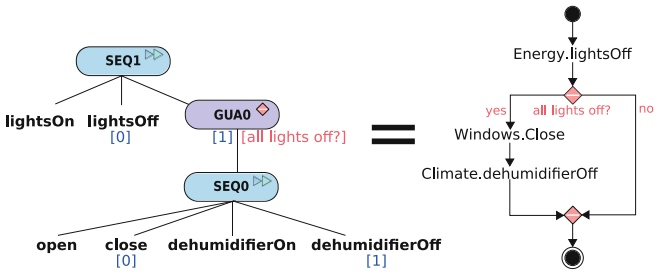

Fig. 16. Choosing a workflow before sleeping.

6 Discussion

We presented a preliminary version of DX-MAN in another paper [3]. In this paper, we present additional semantics that allows the selection of workflows at run-time. We also present a comparison between DX-MAN and current composition approaches in the context of user-centric IoT applications.

Developers can use current composition semantics (e.g., orchestration or choreography) to define a workflow that accommodates as many run-time scenarios as possible. However, it is impossible for them to predict all possibilities during the design-phase and, even if they try, the resulting workflow would potentially require a lot of computing resources, because it becomes larger, more complex and cumbersome as the number of possible scenarios increases. This is

in fact highly likely in IoT applications where the number of available services is always growing.

Although automatic service composition mechanisms could mitigate the ever changing run-time user requirements, their overhead increases exponentially as the number of available services grows [9]. Thus, they are only suitable for a small number of services and straightforward workflows. A large number of services would require a user to wait hours (or even days) before getting a responsive application. For that reason, current automatic composition mechanisms are not yet ready to tackle the imminent scale of user-centric IoT applications.

Even though it is focused on static composition, DX-MAN provides semantics to enable multiple workflows at run-time. In some cases, it may be necessary to change a DX-MAN composition at run-time so as to support even more scenarios. This can be done using automatic composition or dynamic reconfiguration techniques on top of DX-MAN semantics.

In contrast to other composition approaches, DX-MAN does not entail much composition overhead, since there is no need to deploy individual workflows, but only a composite service from which a workflow is chosen (not created) at run-time. In fact, IFTTT or any similar tool can be used on top of DX-MAN to choose a workflow, according to a set of user-defined rules.

At this point, the reader may notice that there are clearly many challenges for future work. We discuss some of them below.

Automatic Service Composition. We believe that our work opens new opportunities for automatic service composition, as this technique can be applied on top of DX-MAN semantics. Services (with all their implicit workflows) can be composed to find more possible workflows at run-time, rather than attempting to construct only one workflow at a time. We are particularly interested in decentralized approaches for automatic service composition, since decentralization is crucial to unleash the full potential of IoT.

Self-adaptive Behaviour. Self-adaptive mechanisms can built on top of DX-MAN to autonomically choose a workflow out of the alternative ones, e.g., based on QoS requirements. A DX-MAN composite service can mutate so as to accommodate changes in the context. However, changing a composition at run-time is not trivial, specially when the response time is critical for the user.

Workflow Validation at Run-Time. As a sequencer connector currently allows the invocation of any operation in any order, there is a need for avoiding invalid sequences (e.g., opening a window three consecutive times). At this stage, it is up to the user to decide which workflows are valid.

Concurrency. DX-MAN only provides support for basic concurrency in parallel invocations. However, many IoT scenarios require active services that can be operating on their own (e.g., using a scheduler). Extending DX-MAN with concurrent capabilities requires further investigation.

Data flows at Run-Time. In DX-MAN, data flow is orthogonal to control flow. Current DX-MAN semantics only allow one data flow for every possible workflow within a composite service. For that reason, at this stage DX-MAN can only be used in scenarios where data flow is unimportant, e.g., actuator triggering. In more complex IoT scenarios, different data flows per workflow will be required. Nevertheless, determining data flows at run-time according to user requirements is a challenging task.

7 Conclusions and Future Work

Users may want to customize their own IoT applications. However, current composition approaches allow the definition of only one workflow at a time. This is not desirable for IoT applications where run-time user requirements are always changing. Although automatic composition is a promising technique to tackle this problem, it is still based on existing composition semantics, thus allowing the definition of only one workflow at a time. For that reason, we need to accommodate run-time user requirements as much as possible during the design phase. In this paper, we presented DX-MAN as a solution for this issue.

The algebraic nature of DX-MAN is suitable to mitigate the impact of change in run-time user requirements. We showed with a small example how DX-MAN allows the definition of (general) composite services that contain multiple workflows. Users only choose the workflow they need out of the alternative ones, rather than resort to the cumbersome and inefficient task of creating individual workflows at run-time.

In the short term, we plan to extend the DX-MAN semantics, in order to enhance the flexibility of composite services. Additionally, as workflows are chosen using JSON documents at this stage, we would like to allow the selection of workflows in a more interactive way (e.g., using a visual tool or voice commands). We are in fact currently working on a visual Web editor to fill this gap.

We believe that DX-MAN opens new research directions to tackle the challenges that user-centric IoT applications pose. Given the novelty of DX-MAN, in what creative ways can you define composite services during the design-phase, in order to accommodate as much as possible run-time user requirements?

References

1. Ahmed, T., Tripathi, A., Srivastava, A.: Rain4Service: an approach towards decentralized web service composition. In: IEEE International Conference on Services Computing (SCC 2014), pp. 267–274 (2014)
2. Arellanes, D., Lau, K.K.: D-XMAN: a platform for total compositionality in service-oriented architectures. In: 7th IEEE International Symposium on Cloud and Service Computing (SC2 2017), pp. 283–286 (2017)
3. Arellanes, D., Lau, K.K.: Exogenous connectors for hierarchical service composition. In: 10th IEEE International Conference on Service Oriented Computing and Applications (SOCA 2017), pp. 125–132 (2017)

4. Arellanes, D., Lau, K.K.: Analysis and classification of service interactions for the scalability of the internet of things. In: IEEE International Congress on Internet of Things (IEEE ICIOT 2018) (2018)
5. Blackstock, M., Lea, R.: WoTKit: a lightweight toolkit for the web of things. In: 3rd International Workshop on the Web of Things (WoT 2012), pp. 1–6 (2012)
6. Brambilla, M., Umuhoza, E., Acerbis, R.: Model-driven development of user interfaces for IoT systems via domain-specific components and patterns. J. Internet Serv. Appl. **8**(1), 14 (2017)
7. Chafle, G., Chandra, S., Mann, V.: Decentralized orchestration of composite web services. In: 13th International World Wide Web Conference (WWW 2004), pp. 134–143 (2004)
8. Cherrier, S., Ghamri-Doudane, Y., Lohier, S., Roussel, G.: D-LITe: distributed logic for internet of things services. In: International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing (ITHINGSCPSCOM 2011), pp. 16–24 (2011)
9. Ciortea, A., Boissier, O., Zimmermann, A., Florea, A.M.: Responsive decentralized composition of service mashups for the internet of things. In: 6th International Conference on the Internet of Things (IoT 2016), pp. 53–61 (2016)
10. Dar, K., Taherkordi, A., Vitenberg, R., Rouvoy, R., Eliassen, F.: Adaptable service composition for very-large-scale Internet of Things systems. In: 11th Middleware Doctoral Symposium (MDS 2011), pp. 1–2 (2011)
11. Ghiani, G., Manca, M., Paternò, F., Santoro, C.: Personalization of context-dependent applications through trigger-action rules. Trans. Comput. Hum. Inter. (TOCHI) **24**(2), 14:1–14:33 (2017)
12. Giang, N.K., Blackstock, M., Lea, R., Leung, V.C.M.: Developing IoT applications in the fog: a distributed dataflow approach. In: 5th International Conference on the Internet of Things (IOT 2015), pp. 155–162 (2015)
13. Glombitza, N., Ebers, S., Pfisterer, D., Fischer, S.: Using BPEL to realize business processes for an internet of things. In: 3rd International Conference on Ad-Hoc Networks and Wireless (ADHOCNETS 2011), pp. 294–307 (2011)
14. Guinard, D., Trifa, V., Wilde, E.: A resource oriented architecture for the Web of Things. In: Internet of Things (IOT 2010), pp. 1–8 (2010)
15. IFTTT: IFTTT (2018). https://ifttt.com/
16. Jaradat, W., Dearle, A., Barker, A.: Towards an autonomous decentralized orchestration system. Concurr. Comput. Pract. Exp. **28**(11), 3164–3179 (2016)
17. Jatoth, C., Gangadharan, G.R., Buyya, R.: Computational intelligence based QoS-aware web service composition: a systematic literature review. IEEE Trans. Serv. Comput. **10**(3), 475–492 (2017)
18. Johnston, W.M., Hanna, J.R.P., Millar, R.J.: Advances in dataflow programming languages. ACM Comput. Surv. **36**(1), 1–34 (2004)
19. Jongmans, S.S., Santini, F., Sargolzaei, M., Arbab, F., Afsarmanesh, H.: Orchestrating web services using Reo: from circuits and behaviors to automatically generated code. Serv. Oriented Comput. Appl. **8**(4), 277–297 (2014)
20. Lau, K.K., Tran, C.M.: X-MAN: an MDE Tool for Component-Based System Development. In: 2012 38th Euromicro Conference on Software Engineering and Advanced Applications, pp. 158–165 (2012)
21. Lau, K.K., Velasco Elizondo, P., Wang, Z.: Exogenous connectors for software components. In: 8th International Conference on Component-Based Software Engineering, pp. 90–106 (2005)
22. Lee, C., Wang, C., Kim, E., Helal, S.: Blueprint flow: a declarative service composition framework for cloud applications. IEEE Access **5**, 17634–17643 (2017)

23. Palomar, E., Chen, X., Liu, Z., Maharjan, S., Bowen, J.: Component-based modelling for scalable smart city systems interoperability: a case study on integrating energy demand response systems. Sensors **16**(11), 1810 (2016)
24. Persson, P., Angelsmark, O.: Calvin – merging cloud and IoT. Procedia Comput. Sci. **52**, 210–217 (2015)
25. Sheng, Q., Qiao, X., Vasilakos, A., Szabo, C., Bourne, S., Xu, X.: Web services composition: a decade's overview. Inf. Sci. **280**, 218–238 (2014)
26. Taušan, N., Markkula, J., Kuvaja, P., Oivo, M.: Choreography in the embedded systems domain: a systematic literature review. Inf. Softw. Technol. **91**, 82–101 (2017)
27. Thuluva, A., Bröring, A., Medagoda Hettige Don, G.P., Anicic, D., Seeger, J.: Recipes for IoT applications. In: 7th International Conference on the Internet of Things (IOT 2017) (2017)
28. Wang, S., Zhou, A., Yang, M., Sun, L., Hsu, C.H., Yang, F.: Service composition in cyber-physical-social systems. IEEE Trans. Emerg. Top. Comput. (2017)
29. Zapier: Zapier — The easiest way to automate your work (2018). https://zapier.com/