Towards a Resource-aware Thing Composition Approach

Zakaria Maamar
Zayed University
Dubai, United Arab Emirates

Saoussen Cheikhrouhou
University of Sfax
Sfax, Tunisia

Muhammad Asim
FAST-NUCES
Islamabad, Pakistan

Ayesha Qamar
FAST-NUCES
Islamabad, Pakistan

Thar Baker
Liverpool John Moores University
Liverpool, UK

Emir Ugljanin
State University of Novi Pazar
Novi Pazar, Serbia

***Abstract*—This paper addresses the silo concern that undermines the participation of IoT-compliant things in composition scenarios. By analogy with composite Web services, each scenario is specified in terms of choreography and orchestration and at design-time and run-time. To define things' execution behaviors during composition, a set of transactional properties known as pivot, retriable, and compensatable, are used allowing to decide when thing execution should be confirmed, rolled-back, or stopped. Along with these properties, another set of availability properties known as limited, renewable, and non-shareable specify the resources that things consume at run-time. Not all resources are always available and hence, could impact the execution of thing composition scenarios. A case study related to Industry 4.0 is used to motivate thing composition.**

***Index Terms*—Choreography, Industry 4.0, IoT, Orchestration.**

I. Introduction

Internet-of-Things (IoT) is one of the fastest growing fields in the ICT world. According to Gartner (www.gartner.com/newsroom/id/3165317), 6.4 billion connected things were in use in 2016, up 3% from 2015, and will reach 20.8 billion by 2020. To tap into the opportunities related to this ever-growing number of things, we recommend engaging things in composition scenarios. Composition is about putting things together so, that, each thing would know what to do, where, when, and with whom to communicate [16]. To specify thing composition, we resort to the concept of process model that underpins the design of any business process.

Despite the tremendous possibilities that IoT offers to the R&D community, most today's things are confined into "silos" due to challenges related to diversity and multiplicity of things' development and communication technologies [3], users' reluctance to and sometimes rejection of things because of privacy invasion [17], limited IoT-platform interoperability [5], lack of an IoT-oriented software engineering discipline that would guide thing analysis, design, and development [23], and, finally, passive nature of things that primarily act as data suppliers (with some actuating capabilities) [19]. Thing composition should address the "silo" concern by allowing things to work together.

By analogy with Web services composition [7], we examine composition of things from 2 perspectives that are *orchestration* (**focus of this paper**) and *choreography*. The former relies on a centralized module that would have full control over all the operations of the component things participating in a composition. And, the latter relies on peer-to-peer interactions that would allow the component things to act freely but, at the same time, be held accountable for their operations' outcomes. Besides these 2 perspectives, we also focus on the execution behavior and resource consumption of things when they participate in compositions. We define the execution behavior with a set of *transactional* properties ([13], e.g., pivot and retriable) and resource consumption with a set of *availability* properties ([15], e.g., limited and renewable). In this paper, we present an approach for composing things that considers the transactional properties of things, the availabilities of resources, and whether orchestration or choreography is adopted.

In what follows Section II presents some related work. Section III motivates thing composition using a case study. Sections IV and V detail the thing composition approach in terms of concepts and operations. Concluding remarks and future work are included in Section VI.

II. Related work

It is largely accepted in the R&D community that today's things are still "primitive" and, that, the lack of universally agreed-upon standards and protocols for thing communication is confining things into separate "silos" refraining their composition [18]. To the best of our knowledge, there are no works dedicated to thing composition in terms of orchestration and choreography. To address this gap, Web services composition can be looked at as a close related domain.

In [20], Asghari et al. discuss a systematic review of 2012-2017 references about service composition in IoT. The review led to developing a technical taxonomy of service composition approaches. For instance, they classified these approaches into functional and non-functional. The former verifies the behavior of compositions such as correctness, reachability, deadlock, and safety. The latter analyzes QoS factors of service composition considering 3 groups of IoT architectures:

infrastructure-based, Internet protocol-based, and application-based. In [6], Chen et al. propose a distributed approach for IoT device management and service composition based on social networks. Heterogeneous IoT devices are encapsulated into Restful Web services so, that, IoT services invocation is made transparent. The authors emphasize the social relations between IoT services to propose a flexible and scalable mechanism that would support IoT services composition. The objective is to satisfy users' requirements by achieving an automatic collaboration of heterogeneous IoT devices. The authors also classify IoT services into location, type, and correlation in order to boost service registration, selection, and management. In [22], Yang and Li propose an IoT information-service composition approach for sensory-data selection and aggregation. The approach consists of user demand analysis, service search and match, service selection, and, finally, service composition. It is based on QoS so, that, an optimal service composition solution is developed using genetic algorithms. In [11], Li and Jiang consider that IoT services are different from traditional services due to the dynamic nature of their environment. They present a context-based approach for IoT service composition and use OWL to build an IoT context ontology. By satisfying certain contextual constraints (i.e., location information, user information, and computing entities), composition meets the needs of service users. Particularly, Li and Jiang's work divides service selection into computational context to select the appropriate services and QoS to select the best services according to users' needs.

Some recent research consider fog in the context of service-oriented resource management model for IoT devices. Wen et al. discuss the emerging issues, challenges, and future research directions in fog enabled orchestration for IoT [21]. Aazam et al. consider the orchestration of IoT networks using fog nodes [1]. And, Andročec proposes a framework to compose things as services and cloud services [2]. He uses ontologies and JSON-LD to semantically annotate services. He also devises a mechanism to semi-automatically compose services.

The existing approaches consider that real-world things provide their functionalities as Web services [1], [6], [9], [11], [12], [20]–[22]. As a result, there is a mix between IoT and service composition, which is not in-line with the separation of concerns principle [10]. Moreover, these approaches are too restrictive since not all connected things can be exposed as services. The impact of resources and transactional properties on thing composition is also not considered. It seems right to put efforts into the development of thing composition approaches that would consider both resources and transactions. This would include defining an architecture for thing composition, addressing both orchestration and choreography issues, managing thing discovery, and handling production/consumption and sharing of resources between things.

III. Case study: IoT-driven Industry 4.0

In line with Industry 4.0 revolution [8], our case study is about a carmaker who embraces IoT and its derivatives (e.g., fog, cloud, cognition, and ubiquity) to improve customer satisfaction, cut operation costs, etc. Tapping into IoT capabilities, several IoT-driven operations could be planned. First, robots that stamp and paint cars constantly report their jobs to a control center. RFID fitted accessories are instantly tracked and dispatched. Welding machines promptly intervene upon the request of heat sensors. Finally, workers communicate and synchronize with peers using wrist devices.

The carmaker would like to increase the business's profit by running his own showrooms for the cars' luxury models (e.g., price above US 50K) instead of relying on third parties. Thus, he would like to oversee the complete sales process from order receipt to car shipping and, finally, car handover. To allow the carmaker to achieve the above benefits, the following thing composition scenarios could be considered:

1) During the order stage: prior to finalizing any order, customers can customize their cars by adding accessories. In that case, the workshop is informed so, that, actions are taken like fine tuning some robots, checking the availability of accessories, and contacting some suppliers. A "smart" warehouse would be of great support allowing to maintain appropriate stock levels along with responding to last-minute requests. Through a dedicated app, the carmaker and customers track the order completion progress thanks to a potential composition of things that would involve smartphones/tablets, cars' bodies, stamping robots, and fixed accessories.
2) During the shipping stage: once the cars are ready for either shipment or direct pickup, "smart" logistics based on sensory networks in specific airports and seaports would permit to track the status of cars in real time, preventing delays and damages. Through the same app, the customers track the locations of their cars and the carmaker sends customs the necessary documents for clearance. A potential composition of things would involve the "smart" warehouse, ships and trucks on the move, customs, and insurance companies.
3) During the handover stage: the same app is used to confirm the agreed upon handover date, to check all the paperwork, and to notify the showroom staff to organize a small ceremony. A potential composition of things would include ready-to-handover cars, the showroom staff, and the road traffic authority.

With respect to order, shipping, and handover stages, some compositions of things are orchestrated while others are choreographed. On the one hand, when the carmaker receives orders, the control center needs to known all these orders since some specialized accessories are expensive and thus, made on-demand. Any delay in their delivery would impact the whole production plan. An orchestrated control of ongoing compositions would offer a complete picture of the progress of each order. This orchestration would aggregate disparate sensor technologies, connect to spare parts' providers, and streamline retail operations with a single view of all customers. On the other hand, when a customer is notified of the car

handover, a choreographed control of ongoing compositions would be enough allowing each participating thing to coordinate its next actions with the peers in the same composition.

IV. Background

As stated in Section I, we either orchestrate or choreograph thing composition. Contrarily, Belkeziz and Jarir mix both to achieve thing coordination and not composition [4]. They argue that existing composition approaches have been geared towards specific case studies and thus, cannot be generalized. Moreover, they note that these approaches align coordination with service orchestration and service choreography, which is not in line with our vision of how things should be composed. Services and things should not be treated at the same level of abstraction. And, techniques for services should not "blindly" be applied to things. Things are "immersed" in cyber-physical surroundings while services are "immersed" in cyber surroundings, only. In this section, we discuss the duties of things and then, present how transactional properties apply to things' duties and how resources are defined so, that, things' duties can consume them.

A. Duties of things

In a previous work [14], we identified things' duties that are *sensing*, *actuating*, and *communicating*. These duties are either enabled or disabled ((0,1) in Fig. 1) according to the requirements that a thing composition is expected to satisfy. We note that thing participation in compositions is strictly dependent on their enabled duties.

From a duty perspective, (*i*) a thing senses the cyber-physical surrounding so, that, it generates (raw) data; (*ii*) a thing actuates data including those that have been sensed; and (*iii*) a thing communicates with the cyber-physical surrounding the data that are sensed and/or actuated. Receiving data and/or commands from external parties (e.g., other things) is also taken care by the communicating duty, but this is not considered in this work.

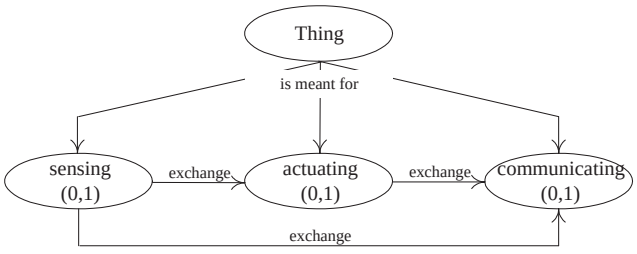

Fig. 1. Representation of a thing's duties

A thing's sensing (s), actuating (a), and communicating (c) duties can be put together which for instance, exemplifies situations of compositions when these duties originate from separate things. The combined duties are as follows (bidirectional "exchange" link in Fig. 1 is not considered in this work, too):

1) s→a→c: sensed data are passed on to actuating; and the data that result from actuating are passed on to communicating for distribution.
2) s→a: sensed data are passed on to actuating; and the data that result from actuating are finals.
3) s→c: sensed data are passed on to communicating for distribution.
4) a→c: data that result from actuating are passed on to communicating for distribution.

B. Transactional properties of things

In the literature, transactional properties are of 3 types [13]: pivot, compensatable, and retriable. These properties are used for different purposes, one of them is to declare when a failed execution is tolerated and when a successful execution can be canceled. The execution concerns task (used hereafter for illustration) in the context of business processes and transaction in the context of databases. A task is pivot if once it successfully completes, its execution effects remain unchanged forever and cannot be undone. Additionally, a pivot task cannot be retried following failure. A task is compensatable if its effects after successful execution can be semantically undone (through another task). And, a task is retriable if it is sure to successfully complete after several finite activations.

In the context of IoT, not all the transactional properties are applicable to things' duties as per the below discussion (Table I where yes/no refer to applicable/not-applicable):

TABLE I
SUMMARY OF COUPLING TRANSACTIONAL PROPERTIES TO A THING'S DUTIES

Transactional property	Thing's duties		
	Sensing	Actuating	Communicating
Pivot	no	no	no
Retriable	yes	yes	yes
Compensatable	no	yes	no

Sensing duty: according to the case study, a car's rain sensor exemplifies the sensing duty.

- Pivot coupled to sensing should not arise; a failed sensing would deprive IoT applications from data.
- Retriable coupled to sensing could arise to guarantee data availability to IoT applications.
- Compensatable coupled to sensing should not arise; a failed sensing is not acceptable (like pivot) and canceling (in the sense of recalling/undoing) what has been sensed is not doable.

Actuating duty: according to the case study, a car's wipers exemplify the actuating duty.

- Pivot coupled to actuating should not arise; a failed actuating would deprive IoT applications from processed/acted upon data.
- Retriable coupled to actuating could arise to guarantee processed/acted upon data availability to IoT applications and/or to allow IoT applications to process/act upon data further.

- Compensatable coupled to actuating could arise; a failed actuating is acceptable and canceling (in the sense of reversing) what has been actuated/acted upon is doable, too.

Communicating duty: according to the case study, a car's GPS exemplifies the communicating duty.

- Pivot coupled to communicating should not arise; a failed communicating would deprive IoT applications from data.
- Retriable coupled to communicating could arise to guarantee data availability to IoT applications.
- Compensatable coupled to communicating should not arise; a failed communicating is not acceptable (like pivot) and canceling (in the sense of recalling/undoing) what has been communicated is not doable.

Out of the above discussion, we conclude that retriable is appropriate for all duties and hence, will be taken into account during thing composition.

C. Resource availabilities to things

To define the availability properties of resources so, that, things plan the consumption of these resources, we build upon our previous work on social coordination of BPs [15] and consider the following properties:

- limited (l): the consumption of a resource is bound to a quantity and/or time period. On the one hand, the resource (e.g., 2 paint jars) becomes unavailable for all things once the quantity is finished. On the other hand, the resource (e.g., robots' weekly service plan) becomes unavailable for a particular thing beyond the time period, but could remain available for other things for consumption, should these things be not concerned with the time period.
- Renewable (rn): contrarily to limited, a thing could extend the consumption of a resource for another time period (e.g., warehouse access card extended for another month) or request an additional quantity (e.g., 1 additional paint jar), should the resource remain available (e.g., not requested by another thing and not-violating any constraint).
- Non-shareable (ns): when the concurrent consumption of a resource (e.g., vehicle certificate document) by many things needs to be scheduled.

Unless stated, we assume that a resource is by default unlimited (ul) and/or shareable (s). To charge resource consumption, we develop 3 categories of prices that are commonly used in the airline industry for economy tickets (e.g., emirates.com) (Table II): saver, flex, and flex+. Each category promotes a different refund and change policy prior to consuming resources. It is worth noting that prices (e.g., saver) for some resources might not be available or sold-out at the discretion of their providers. In Table II, refund and change policies target things (i.e., owners) that could consider canceling/postponing their resource consumption, selecting other resources after agreeing on others, and/or revising their demands of resources.

- Saver is the lowest price due to no refund and no change.

TABLE II
PRICE BREAKDOWN FOR A RESOURCE

Policy	Prices		
	saver: lowest	flex: in-between	flex+: highest
Refund	No refund	With a fee	No fee
Change	No change	With a fee	No fee

- Flex+ is the highest price due to refund and change at no-cost.
- Flex is between saver and flex+ prices due to refund and change with a fee.

V. ORCHESTRATION-DRIVEN, RESOURCE-AWARE COMPOSITION OF THINGS

Our objective is to check the **doability** of having a composition of things with respect to (*i*) composition types (orchestration *versus* choreography), (*ii*) transactional properties (pivot *versus* retriable *versus* compensatable), and (*iii*) availability properties (limited *versus* renewable *versus* non-shareable). It happens that a composition cannot occur at run-time due to certain transactional and/or availability properties. We note that thing description, discovery, and selection for participation in compositions do not fall into this paper's scope.

In compliance with Table I, only retriable property is applicable to all duties of a thing. As a result, pivot and compensatable properties are discarded from our resource-aware composition approach. For a retriable thing, resources are required to secure its first-time execution and, in the case of failure, to retry its (re-)execution until success. Ideally, a thing would like to use same resources for both first execution and re-execution. However, this might not always be possible during re-execution as some resources could "vanish" being consumed by or assigned to other things at run-time. One assumption is that any resource assigned to a thing covers one execution cycle of this thing, whether this cycle leads to success or failure.

A. Definitions

Definition 1: We define a **resource r** as a tuple $< id, desc, ava, P >$ where: id is a resource identifier; $desc$ is a textual description of the resource (either compute, store, or communicate); ava is an availability property of the resource ($ava \in \{\mathsf{uls}, \mathsf{l}, \mathsf{r}, \mathsf{ns}\}$); unless stated, a resource is unlimited and shareable ($ava = \mathsf{uls}$); and, P is a set of prices of the resource according to its availability property and consists of 3 couples $< cat, p >$ where cat is a price category of the resource ($cat \in \{\mathsf{saver}, \mathsf{flex}, \mathsf{flex+}\}$) and p is a positive value for a particular price category ($p \in \mathbb{R}^+$). It happens that p is zero referring to the unavailability of a price category.

Example: $\mathbf{res}_1 = < id_1, compute, \mathsf{r}, \{< \mathsf{saver}, \$2 >, < \mathsf{flex}, \$3 >, < \mathsf{flex+}, \$0 >\} >$ reads as follows; $\mathbf{r}_1$, whose identifier is id_1, is a resource of type $compute$, has a renewable availability r, and is offered in 2 different prices: \$2 as a saver price and \$3 as a flex price. $\mathbf{r}_1$ is not offered as a flex+ price.

Definition 2: We define a **thing t** as a tuple $< id, desc, D, R >$ where: id is a thing identifier; $desc$ is a

textual description of the thing; D is a couple $< d, t_p >$ where d is a (isolated/combined) duty ($d \in \{\mathsf{s}, \mathsf{a\,c}, \mathsf{sa}, \mathsf{sc}, \mathsf{ac}, \mathsf{sac}\}$) that the thing performs along with a transactional property t_p ($t_p \in \{\mathsf{pivot}, \mathsf{retriable}, \mathsf{compensatable}\}$) of this duty; and R is a set of resources $\{r\}$ assigned to the thing for consumption. **Example:** $\mathbf{t}_1 = < id_1, sensor, < \mathsf{s}, retriable >, \{r_1, r_i\} >$ reads as follows; $\mathbf{t}_1$, whose identifier is id_1, is a thing of type $sensor$, has sensing s as a duty that is retriable, and consumes 2 resources that are r_1 and r_i. Since a resource could have up to 3 prices, the price to select will be worked out at run-time. For the time being, the 3 prices are considered.

Definition 3: We define a **thing composition schema** $\mathcal{C}_\mathcal{T}$ as an oriented acyclic graph (T, A) where: T is a set of nodes that correspond to things $\{t_1, t_2, ..., t_n\}$; and $A \leftarrow T \times T$ is a set of ordered pairs of arcs (t_i, t_j). (t_i, t_j) represents an execution dependency so, that, t_i is invoked before t_j.

B. Thing orchestration

Fig. 2 represents an orchestration-based thing composition where a central module, orchestrator, coordinates the sequence of (could be concurrent) things to invoke during composition completion at run-time. In the same figure, a pool of resources is made available to all things in terms of consumption $(0, n)$ and production $(0, n)$. The second case is not considered for the time being (i.e., things do not produce resources).

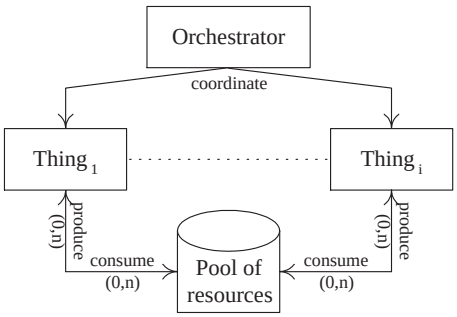

Fig. 2. Thing composition as an orchestration

Being a central element, the orchestrator is aware of the resources to consume by all things, the transactional properties of all things, and the availability properties of all resources. We develop thing orchestration along 2 stages: design time and run time.

1) Design-time stage: We designed Algorithm 1 that takes 2 inputs, $\mathcal{C}_\mathcal{T}$ and $\mathcal{R}_{\mathcal{C}_\mathcal{T}}$ as a resource-to-consume set. It includes all resources to complete $\mathcal{C}_\mathcal{T}$. $\mathcal{R}$ is the set of all resources available in the IoT ecosystem, and produces 1 output, a set $\mathcal{G}$ (line 2) that includes all *clusters* (g_{r_j}, line 4) of things in $\mathcal{C}_\mathcal{T}$ consuming same resources $r_j \in \mathcal{R}_{\mathcal{C}_\mathcal{T}}$.

The algorithm checks things in $\mathcal{C}_\mathcal{T}$ one by one (line 6) along with checking the resources in $\mathcal{R}_{\mathcal{C}_\mathcal{T}}$ one by one, too (line 7). The objective is to form one resource-wise *cluster*, i.e., g_{r_j}, per resource that same things will consume. In line 8, $cons$ function checks if a thing t_i is linked to a resource r_j. If yes, t_i is added to g_{r_j} (lines 9-24). Otherwise, t_i is added to $g_{r_{k \neq j}}$ (g_{r_k} already exists or will be created).

In terms of prices of a resource, 2 cases are possible depending on the resource's availability property.

- Should the resource r_j be limited (line 9), saver is recommended first followed by flex then flex+. Although saver is the lowest, it could happen that t_i ends up paying more if its execution fails and hence, needs to be re-executed while considering that r_j could become unavailable (i.e., limited) or assigned to another thing. This will be found out at run-time.
- Should the resource r_j be renewable (line 20), flex is recommended first then flex+. If t_i fails, it can still request consuming r_j with a minimal fee since flex allows to extend the use of r_j assuming it is still available or not-assigned to another thing. This will be found out at run-time.

At the end of running the algorithm, $\mathcal{G}$ is returned based on all g_{r_j} (lines 31-34).

2) Run-time stage: For run-time orchestration, detailed in Algorithm 2, the set of all resource *clusters* $\mathcal{G}$ will come initialized from design-time orchestration having all things $\{t_{i=1...m}\} \in \mathcal{C}_\mathcal{T}$ allocated to the *clusters* $\{g_{r_j}\}$ with $\{t_i\} \in \{g_{r_{j=1...n}}\}$ (a thing could belong to many *clusters*). We launch run-time orchestration algorithm by checking all things t_i, whose allocated resources r_j are already set at design-time (at least one resource for one thing; $j \geq 1$). For the sake of simplicity, a resource is always available to all things guaranteeing first-time execution of these things. The different allocations will be updated according to run-time scenarios as discussed in Algorithm 2:

VI. Conclusion

Developing advanced IoT applications to respond to users' complex and changing needs, calls for new techniques that would allow for instance, to engage things in composition scenarios. In this paper we presented our on-going research on thing composition taking into account the type of composition (orchestration *versus* choreography), the transactional properties of things (pivot *versus* retriable *versus* compensatable), and the availability properties of resources (limited *versus* renewable *versus* non-shareable) that things consume at run-time. We motivated thing composition using a case study about Industry 4.0. In this paper, we mainly focussed on the conceptual work of thing composition by presenting orchestration-based composition algorithms. Our ongoing work is to complete choreography-based composition algorithms and implement all algorithms using real things.

Acknowledgment

The authors would like to thank Mohamed Sellami and Slim Kellal for their initial contributions to the work presented in this paper.

Algorithm 1 Design-time thing orchestration

procedure ALLOCORCHESTRATION($\mathcal{C}_\mathcal{T}$, $\mathcal{R}_{\mathcal{C}_\mathcal{T}}$, $\mathcal{G}$)
  $\mathcal{G} \leftarrow$ null ▷ Initialize $\mathcal{G}$
  for all $r_j \in \mathcal{R}_{\mathcal{C}_\mathcal{T}}$ **do**
    $g_{r_j} \leftarrow$ null ▷ Initialize $clusters$
  end for
  for all $t_i \in \mathcal{C}_\mathcal{T}$ **do** ▷ check things one by one in $\mathcal{C}_\mathcal{T}$
    for all $r_j \in \mathcal{R}_{\mathcal{C}_\mathcal{T}}$ **do** ▷ check resources one by one in $\mathcal{R}_{\mathcal{C}_\mathcal{T}}$
      if $cons(t_i, r_j)$ **then** ▷ does t_i consumes r_j?
        if ($r_j.a =$ l) **then** ▷ r_j is limited
          if (saver $\in r_j.p$) **then**
            $g_{r_j} \leftarrow g_{r_j} \cup <t_i$,saver>
          else
            if (flex $\in r_j.p$) **then**
              $g_{r_j} \leftarrow g_{r_j} \cup <t_i$,flex>
            else
              $g_{r_j} \leftarrow g_{r_j} \cup <t_i$,flex+>
            end if
          end if
        else
          if ($r_j.a =$ r) **then** ▷ r_j is renewable
            if (flex $\in r_j.p$) **then**
              $g_{r_j} \leftarrow g_{r_j} \cup <t_i$,flex>
            else
              $g_{r_j} \leftarrow g_{r_j} \cup <t_i$,flex+>
            end if
          end if
        end if
      end if
    end for
  end for
  for all $r_j \in \mathcal{R}_{\mathcal{C}_\mathcal{T}}$ **do**
    $\mathcal{G} \leftarrow \mathcal{G} \cup g_{r_j}$ ▷ Set of *clusters* of all resources
  end for
  return $\mathcal{G}$
end procedure

REFERENCES

[1] M. Aazam and E. Huh. Fog Computing Micro Datacenter Based Dynamic Resource Estimation and Pricing Model for IoT. In *Proceedings of the IEEE 29th International Conference on Advanced Information Networking and Applications (AINA'2015)*, South Korea, March 2015.

[2] D. Androcec. Using JSON-LD to Compose Different IoT and Cloud Services. *CoRR*, abs/1809.08233, 2018.

[3] D. Androcèec, B. Tomaà, and T. Kièasondi. Interoperability and Lightweight Security for Simple IoT Devices. In *Proceedings of the Information Systems Security Conference (ISS'2017) held in conjunction with the 40th Jubilee International Convention on Information and Communication Technology, Electronics, and Microelectronics (MIPRO'2017)*, Opatija, Croatia, May 2017.

[4] R. Belkeziz and Z. Jarir. IoT Coordination: Designing a Context-Driven Architecture. In *Proceedings of the 13th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS'2017)*, Jaipur, India, 2017.

[5] A. Bröring, A. Ziller, V. Charpenay, S. Schmid, A.S. Thuluva, D. Anicic, A. Zappa, M.P. Linares, L. Mikkelsen, and C. Seidel. The BIG IoT API - Semantically Enabling IoT Interoperability. *IEEE Pervasive Computing*, August 2018 (forthcoming).

[6] G. Chen, J. Huang, B. Cheng, and J. Chen. A Social Network Based Approach for IoT Device Management and Service Composition. In *Proceedings of the IEEE World Congress on Services (SERVICES'2015)*, New York, USA, June 2015.

[7] J. Han, Y. Luo, and J. Huang. A Study on the Scalable Flow Model of Web Services Choreography and Orchestration based on Dynamic Workflow. *International Journal of Information and Communication Technology*, 5(3/4), 2013.

[8] L. Heiner, F. Peter, K. Hans-Georg, F. Thomas, and H. Michael. Industry 4.0. *Business & Information Systems Engineering*, 6(4), 2014.

[9] A. Kouicem, A. Chibani, A. Tari, Y. Amirat, and Z. Tari. Dynamic Services Selection Approach for the Composition of Complex Services in the Web of Objects. In *Proceedings of the IEEE World Forum on Internet of Things (WF-IoT'2014)*, Seoul, South Korea, March 2014.

[10] H. F. Ledgard and Y. Kambayashi. The Separation Principle: A Programming Paradigm. *IEEE Software*, 21, 2004.

[11] K. Li and L. Jiang. The research of Web Services Composition based on Context in Internet of Things. In *Proceedings of the IEEE International Conference on Computer Science and Automation Engineering (CSAE'2012)*, volume 1, Shanghai, China, May 2012.

[12] L. Li, Z. Jin, G. Li, L. Zheng, and Q. Wei. Modeling and Analyzing the Reliability and Cost of Service Composition in the IoT: A Probabilistic Approach. In *Proceedings of the IEEE 19th International Conference on Web Services (ICWS'2012)*, Honolulu, HI, USA, June 2012.

[13] M. Little. Transactions and Web Services. *Communications of the ACM*,

Algorithm 2 Run-time thing orchestration

1: **procedure** ALLOCORCHESTRATION($\mathcal{C}_\mathcal{T}$, $\mathcal{R}_{\mathcal{C}_\mathcal{T}}$, $\mathcal{G}$)
2: **for all** $t_i \in \mathcal{C}_\mathcal{T}$ **do** ▷ Visit all things in the composition
3: **for all** g_{r_j} to which t_i belongs **do** ▷ Visit all *clusters* where t_i is included
4: $execute(t_i)$
5: **if** $fail(t_i)$ **then**
6: **if** $allowRetrial(t_i)$ **then** ▷ Check if t_i can re-execute with respect to threshold
7: **if** $ava(r_j)$ **then** ▷ Check if r_j is available
8: $reallocate(r_j, t_i)$ ▷ r_j is assigned again to t_i
9: Go to line 4
10: **else** ▷ r_j is no longer available
11: $g_{r_j} \leftarrow g_{r_j} - t_i$ ▷ Drop t_i from g_{r_j}
12: **if** ($search(r_{j'}, \{\mathcal{R}_{\mathcal{C}_\mathcal{T}} - r_j\}$ or resource pool)= true) **then** ▷ Check for another resource $r_{j'}$
13: $g_{r_{j'}} \leftarrow g_{r_{j'}} \cup < t_i\ ,\ optimal(r_{j'}.p) >$ ▷ Take optimal available price of $r_{j'}$
14: **if** $g_{r_{j'}}$ $is\ currently\ created$ **then**
15: $\mathcal{G} \leftarrow \mathcal{G}\ \cup g_{r_{j'}}$ ▷ Update $\mathcal{G}$
16: **end if**
17: $execute(t_i)$
18: **break**
19: **else** ▷ No alternative $r_{j'}$ exists in $\mathcal{R}_{\mathcal{C}_\mathcal{T}}$ and resource pool
20: stop ▷ Composition of t_i fails
21: **end if**
22: **end if**
23: **else**
24: stop ▷ t_i not allowed to retry; composition fails
25: **end if**
26: **end if**
27: **end for**
28: **end for**
29: **end procedure**

46(10), October 2003.

[14] Z. Maamar, T. Baker, M. Sellami, M. Asim, E. Ugljanin, and N. Faci. Cloud *versus* Edge: Who Serves the Internet-of-Things Better? *Internet Technology Letters, Wiley*, June 2018 (https://tinyurl.com/y767ybor).

[15] Z. Maamar, N. Faci, S. Sakr, M. Boukhebouze, and A. Barnawi. Network-based Social Coordination of Business Processes. *Information Systems*, 58, 2016.

[16] Z. Maamar, M. Sellami, N. Faci, E. Ugljanin, and Q.Z. Sheng. Storytelling Integration of the Internet of Things into Business Processes. In *Proceedings of the Business Process Management Forum (BPM Forum'2018) held in conjunction with the 16th International Conference on Business Process Management (BPM'2018)*, Sydney, NSW, Australia, 2018.

[17] A. Martínez-Ballesté, P.A. Pérez-Martínez, and A. Solanas. The Pursuit of Citizens' Privacy: a Privacy-Aware Smart City is Possible. *IEEE Communications Magazine*, 51(6), 2013.

[18] Radu-Casian Mihailescu, Romina Spalazzese, Clint Heyer, and Paul Davidsson. A Role-Based Approach for Orchestrating Emergent Configurations in the Internet of Things. *CoRR*, abs/1809.09870, 2018.

[19] A.M. Mzahm, M.S. Ahmad, and A.Y.C. Tang. Agents of Things (AoT): An intelligent operational concept of the Internet of Things (IoT). In *Proceedings of the 13th International Conference on Intellient Systems Design and Applications (ISDA'2013)*, Bangi, Malaysia, 2013.

[20] A. Parvaneh, R. Amir Masoud, and J. Hamid Haj Seyyed. Service Composition Approaches in IoT: A systematic review. *Journal of Network and Computer Applications*, 120, 2018.

[21] Z. Wen, R. Yang, P. Garraghan, T. Lin, J. Xu, and M. Rovatsos. Fog Orchestration for IoT Services: Issues, Challenges and Directions. *IEEE Internet Computing*, 21(2), 2017.

[22] Z. Yang and D. Li. IoT Information Service Composition Driven by User Requirement. In *Proceedings of the IEEE 17th International Conference on Computational Science and Engineering (CSE' 2014)*, Chengdu, China, Dec 2014.

[23] F. Zambonelli. Key Abstractions for IoT-Oriented Software Engineering. *IEEE Software*, 34(1), January-February 2017.