Allocation of Resources in SAaaS Clouds Managing Thing Mashups

J. Guerreiro, L. Rodrigues and N. Correia

***Abstract*—The sensing and actuation as-a-service is an emerging business model to make sensors, actuators and data from the Internet of Things more attainable to everyday consumer. With the increase in the number of accessible Things, mashups can be created to combine services/data from one or multiple Things with services/data from virtual Web resources. These may involve complex tasks, with high computation requirements, and for this reason cloud infrastructures are envisaged as the most appropriate solution for storage and processing. This means that cloud-based services should be prepared to manage Thing mashups. Mashup management within the cloud allows not only the optimization of resources but also the reduction of the delay associated with data travel between client applications and the cloud. In this article, an optimization model is developed for the optimal allocation of resources in clouds under the sensing and actuation as-a-service paradigm. A heuristic algorithm is also proposed to solve the problem more quickly.**

***Index Terms*—Internet of Things, Cloud, Sensing and actuation as-a-service, Heuristic.**

I. Introduction

THE Internet of Things (IoT) is now attracting the attention from both academia and industry, and this interest is expected to grow [1], [2]. In IoT, physical objects can be accessed and controlled using electronic devices that are able to communicate using networking interfaces. However, the research in IoT is primarily driven by technological advances and not by applications or user needs. On the other hand, research on smart cities, smart transportation, and others, address specific problems and needs citeCetal19. An effective bridge between these two relies on an efficient resource discovery, access and management, which can be provided by what is now called the Web of Things (WoT) [4].

A move towards the WoT will prevent IoT from becoming just a collection of Things, unable to be discovered for interaction with other Things or applications. The idea of WoT is to reuse and leverage readily available and widely popular Web protocols, standards and blueprints, to make data and services offered by objects accessible to a larger pool of Web developers. Things expose their functionality and properties as Web resources, allowing reading (e.g., temperature value) and/or update (e.g., trigger an actuation) by others [5]. This means that any kind of behaviour can be implemented through Web resources. Besides discovery, such Thing exposure will facilitate the creation of mashups, where services/data from one or multiple Things are combined with services/data from virtual Web resources (e.g., multiple sensor data sources can be combined with virtual Web resources to decide for an actuation at some device). As stated in [5], the WoT is intended to enable interoperability across IoT platforms and application domains.

The just mentioned developments will bring many different devices into the IoT world, and large amounts of data will be collected and analysed. As more and more Things become available, and Thing mashups are built, more data with processing needs will emerge, meaning that new challenges arise in terms of storage and processing. To deal with these issues, a Sensing and Actuation as-a-Service (SAaaS) model relying on cloud infrastructures is proposed in [6]. Applications are assumed to have software components with bindings to virtual Things stored at the cloud, creating a multi-user environment assisting in the use of resource-constrained devices.

The work in [6] is relevant because it introduces the idea of Things as infrastructure for cloud-like exploitation. The authors envision sensing and actuation resources not only as mere data endpoints, and instead propose their abstraction, virtualization, and their administration in groups. The architecture and required modules are also discussed. However, there is no reference on how to perform many-to-one assignments (single virtual Thing serving multiple compatible requests), which requires deciding on the best assignments. Since a virtual Thing is then materialized onto a physical Thing, this is a way for a single device to serve multiple consumers. Also, although the possibility to manage Thing mashups is mentioned, no details are given. Managing Thing mashups at the cloud, instead of leaving this to the client, has significant advantages: i) multiple data travels to the client side are avoided, improving delay and Quality of Service (QoS); ii) mashups involve workflow between mashup elements, and the assignment of mashup elements to virtual Things (just mentioned many-to-one

J. Guerreiro, L. Rodrigues and N. Correia are with the Center for Electronic, Optoelectronic and Telecommunications (CEOT).

N. Correia (email: ncorreia@ualg.pt) is with the Faculty of Science and Technology, J. Guerreiro and L. Rodrigues are researchers (emails: {jdguerreiro,lrodrig}@ualg.pt), all at University of Algarve, 8005-139 Faro, Portugal.

assignments) should take such flow dependencies into account, for optimization of flows between workspaces of virtual Things at the cloud. This requires the development of new resource allocation approaches, which has been addressed recently in [7]. In such work, a theoretical model is proposed and used as a basis for the development of a heuristic. However, the model just outlines the problem and is not formulated to allow the extraction of the optimal solution. The developed heuristic has also drawbacks, as detailed in Section V. Here in this article, a new approach to solve this problem is proposed. The contributions of this article are the following:

- A mathematical programming optimization model is developed for resource allocation under the SAaaS paradigm, considering Thing mashups managed in the cloud. This optimization model can be solved by software optimizers like CPLEX[1] for the optimal solution to be extracted.
- A heuristic algorithm is proposed that is able to obtain near optimal solutions quickly, which is critical when problem instances are large.
- The proposed optimization model and heuristic algorithm are compared against the results presented in [7]. The optimization model is also used to assess the benefits of managing Thing mashups in the cloud, when compared with their management at the client side.

The remainder of this article is organized as follows. In Section II, the design and planning of SAaaS architectures is discussed. Related work is presented in Section III. Section IV formulates the resource allocation optimization problem and Section V discusses a heuristic algorithm to solve it. Section VI makes a performance analysis, and Section VII concludes the article.

II. Sensing and Actuation as-a-Service

A. Everything as a Service

The Everything as a Service (XaaS) includes a set of service models under the paradigm of cloud computing that aim to concentrate software and hardware resources, offering them as services to a large number of users and, therefore, leveraging utility and consumption of computing resources. The most relevant service models are: i) Infrastructure as a Service (IaaS), where computing resources like virtual machines, servers, storage and load balancers are provided according to customer requirements; ii) Platform as a Service (PaaS), where computing platforms including operating system, programming language execution environment, database, Web server, and other, are provided in a way that the user is not required to allocate resources manually; iii) Software as a Service (SaaS), where the cloud takes over the infrastructure and platform while scaling automatically [8].

[1]IBM ILOG CPLEX Optimizer.

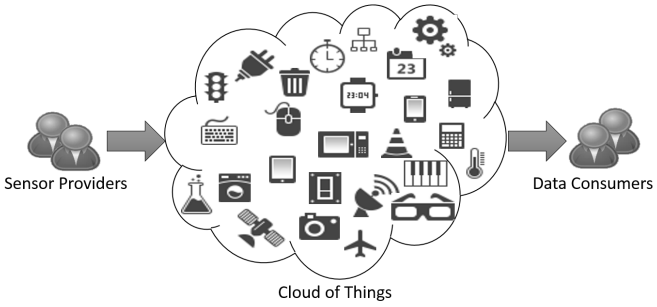

Fig. 1. SAaaS scenario.

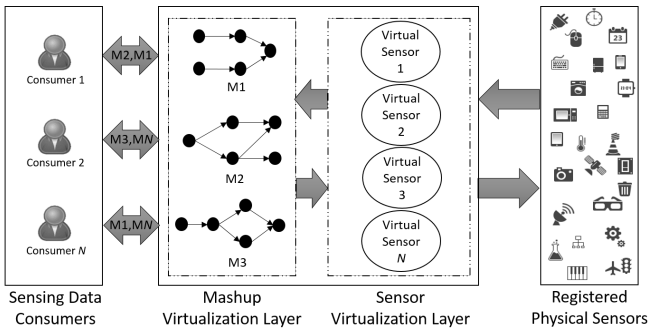

Fig. 2. SAaaS virtualization with mashups managed in the cloud.

The just mentioned models promote the "pay only for what you use" while allowing companies to focus on their core competencies instead of ICT [2], [9]. The SAaaS model, proposed in [6], emerged more recently to assist in the use of resource-constrained Things, as illustrated in Figure 1. Similarly to the just mentioned cloud-based "as a service" models, the resources in SAaaS systems should be dynamically provisioned and de-provisioned on demand. The virtualization of Things is used to enable the management and customization of devices by clients/applications/consumers, eventually allowing for the assignment of a single device to multiple consumers. A virtual workspace (e.g., virtual machine) is usually created for the provisioning of a virtual Thing group (one or more virtual Things), which can be under the control of one or more consumers.

When Thing mashups are managed in the cloud, the events are processed and actuations are triggered according to a workflow that is predefined by the client application. The cloud delivers just the final data of interest to the consumer/client application. The whole mashup, or parts of it, may also be consumed by multiple applications. This additional system functionality results in an additional virtualization layer, as illustrated in Figure 2. Managing Thing mashups in the cloud brings new challenges regarding resource assignment (both physical Thing and cloud resources). More specifically, each mashup ends up defining flow dependencies between its mashup elements, and these should be taken into account when assigning [7]: i)

one or more mashup elements (from different client applications) to a virtual Thing; ii) virtual Things to physical Things (materialization onto devices). This awareness will allow the optimization of both physical Things and cloud resources, ensuring a minimization of the number of virtual workspaces and number of flows between such virtual workspaces. This issue is addressed here in this article.

III. Related Work

Over the last years, cloud-based platforms became popular for Wireless Sensor Network (WSN) applications. Due to limitation in memory, energy, computation and scalability, large WSNs are difficult to manage and for this reason their integration with the cloud is proposed in [10]. The authors suggest a shift from traditional WSNs to cloud-based architectures, and a virtualization model is presented for a uniform and widespread use of WSNs. The authors conclude that sensor-cloud architectures outperform traditional WSNs, allowing the increase of sensor lifetime, decrease of energy consumption, and reduction of end user expenditure. Such WSN-cloud integration can become more efficient if approaches like the ones mentioned in [11], [12], [13], [14] are used. In [11], a sustainable way of collecting data from WSNs to the cloud is proposed. More specifically, the authors state that the weak communication ability of WSNs can make the upload (to the cloud) of big sensed data, within a limited time, quite difficult. Sensors have also limited power, and this kind of data transfer significantly shortens the lifetime of WSNs. To solve these problems, multiple mobile sinks are proposed to help in data collection. In [12] it is proposed that gateways find anomalies in the sensed values, reducing storage requirements if these are discarded, while in [13] a mechanism is proposed that allows sensor nodes to save their energy due to the aggregation of application requests by the cloud. The use of IaaS paradigm, to enhance the flexibility and scalability of such architectures, is proposed in [14].

More recently, cloud-based Sensing as-a-Service (SeaaS) approaches started to emerge. This kind of service is discussed in [2] and [15] in the context of smart cities. The first work addresses technological, economical and social perspectives, while the last proposes the abstraction of physical Things through semantics, so that these can be integrated by neglecting their underlying architecture. In [16] and [17], the semantic selection of sensors is also addressed. Multimedia SeaaS is explored in [18], [19], [20], [21], and the focus is mainly real-type communication requirements. In [20], the cloud edge and fog are explored. Mobile CrowdSensing (MCS) services using cloud infrastructures, which take into account the mobility of data acquisition systems, are also studied in [22] and [23]. The general idea is to use the sensors of mobile devices to fulfill some need. In [24], for example, crowdsourced life streaming, where individuals become online broadcasters, is considered. These are basically mobile SeaaS approaches. Specific platforms providing efficient sharing mechanisms for data (among multiple applications) are proposed in [25], [26].

Instead of considering just sensing devices, some proposals comprise an IoT including smart Things. Besides sensing, smart Things may perform tasks, like actuation and control, and are less focused on pure data gathering, like sensor networks [27]. Approaches for smart objects must consider the functionality of devices, instead of just focusing on the data [6], [7]. Physical resources must be abstracted, virtualized, and presented as a service to the end users. This way, the access and interaction with physical Things becomes uniform and in compliance with IoT/WoT goals. In [6], the design and technical aspects of such cloud-based Sensing and Actuation as-a-Service (SAaaS) architectures are discussed. A specific platform is proposed in [28], which is basically an extension of [25].

Mashup tools to connect smart Things have also been proposed, most of them to be used at the application/client side [29], [30], [31], [32]. Thing mashup management in the cloud is considered in [25], and in [33] IoT mashups as a service is proposed. Discussion is, however, around concepts and architecture of the service model, and no specific approach for resource allocation is proposed. As far as known, this has only been addressed in [7]. Table I, presents the resource allocation strategies (RAS) limitations in the related work articles in comparison to our work: Table I presents the resource allocation strategies (RAS) limitations in the related work articles in comparison to our work:

Here in this article, and similarly to [7], resource allocation in clouds managing Thing mashups is addressed. A mathematical programming optimization model is proposed that allows optimal solutions to be obtained, which is not possible with the approach from [7]. A heuristic algorithm is also proposed that outperforms the results obtained in [7].

IV. Resource Allocation Problem

A. Assumptions and Definitions

A Cloud Service Provider (CSP), denoted by $\mathcal{S}$, includes a set of distributed networks that interconnect to provide services, and these can be organized according to a common role or in order to better serve certain regions. Therefore, $\mathcal{S} = \{\mathcal{S}_1, ...\mathcal{S}_{|\mathcal{S}|}\}$, where $\mathcal{S}_i$ denotes one of the distributed networks. The set of all client applications outside the cloud, and requesting for registered physical Things, is denoted by $\mathcal{A} = \{\mathcal{A}_1, ..., \mathcal{A}_{|\mathcal{A}|}\}$. An application $\mathcal{A}_i$ can have one or more independent components, denoted by $\mathcal{C}(\mathcal{A}_i) = \{\mathcal{C}^i_1, ..., \mathcal{C}^i_{|\mathcal{C}(\mathcal{A}_i)|}\}$, and each component $\mathcal{C}^i_j$ is binded to a Thing mashup in the cloud.

TABLE I
RESOURCE ALLOCATION STRATEGIES LIMITATIONS

Authors	Limitations
S.Misra et al[10]	No RAS to determine properties closer to requests. Does not use mashups embedded in the cloud.
T.Wang et al[11]	No RAS is presented in the paper.
L.Kumar et al[12]	No RAS is presented in the paper.
T.Dinh et al[13]	Does not use mashups embedded in the cloud.
A.Deshwal et al[14]	RAS satisfies one by one request. Does not use clusters or mashups on cloud side.
C.Perera at al[2]	No RAS is presented in the paper.
R.Petrolo at al[15]	No RAS is presented in the paper.
S.Misra et al[16]	RAS using a broker to publish/subscribe data. No RLS to determine properties closer to requests. Does not use clusters or mashups on the cloud side.
Y.Hsu et al[17]	Semantic RAS satisfies one by one request. No RLS to determine properties closer to requests. Does not use mashups on the cloud side.
C.Lai et al[18]	No RAS to determine properties closer to requests. Does not use mashups embedded in the cloud.
C.Lai et al[19]	No RAS to determine properties closer to requests. Does not use clusters or mashups on cloud side.
W.Wang et al[20]	RAS using energy cost estimation as basis. Does not consider data flows or delays in the cloud. No RAS to determine properties closer to requests. Does not use mashups embedded in the cloud.
Y. Xu et al[21]	No RAS is presented in the paper.
X.Sheng et al[22]	Does not use clusters or mashups on cloud side. RAS using phone location. Use only mobile phone services.
M.Al-Fagih et al[23]	RAS with delay and quality requirements. Does not consider processing or network delays. No RAS to determine properties closer to requests. Does not use mashups embedded in the cloud.
C.Dong et al[24]	Time slot RAS for video distribution. Does not use clusters or mashups on cloud side.
J.Kim et al[25]	RAS chooses devices/mashups to collect data. No RAS to determine properties closer to requests.
M.Kim et al[26]	RAS where users subscribe from a list of sensors. No RAS to determine properties closer to requests. Does not use mashups embedded in the cloud.
E.Al-Hawri et al[27]	No RAS is presented in the paper.
S.Distefano et al[6]	Semantic RAS to map devices to requests. No RAS to determine properties closer to requests. Does not use mashups embedded in the cloud.
J.Guerreiro et al[7]	Predecessor article of this work. Some drawbacks explained in section .
R.Casadei et al[**?**]	No RAS is presented in the paper.
F.Longo et al[28]	No RAS to determine properties closer to requests. Does not use mashups embedded in the cloud.
R.Kleinfeld et al[**?**]	No RAS to determine properties closer to requests.
H.Oh et al[29]	No RAS to determine properties closer to requests.
S.Heo et al[30]	No RAS to determine properties closer to requests.
X.Jin et al[31]	No RAS to determine properties closer to requests.
S.Eom et al[32]	No RAS to determine properties closer to requests.
D.Guinard et al[**?**]	No RAS to determine properties closer to requests.
J.Im et al[33]	No RAS to determine properties closer to requests.

In general, a mashup can be defined as a way to compose a new service from existing services [34]. The focus is on mashing up information services. However, with the recent efforts on WoT standardization by W3C (see [5]), Things will also be able to expose their functionality and properties as Web resources, allowing Thing mashups to be built using existing Web mashup technologies. Semantic search of Things is also considered in [5]. For the particular case of SAaaS, clients will be using templates to draw Thing mashups that integrate Things with services/data from virtual Web resources. When drawing such mashups, Things will be mashup elements having a functionality requirement and property conditions, which should be semantic-based (see [35]).

Definition 1 (Thing Mashup)**.** *Workflow wiring together Things with virtual Web resources. When drawing the mashup, Things will be mashup elements with a functionality requirement and a set of property conditions. The functionality of mashup element n is denoted by f_n, while $\mathcal{P}_n$ denotes its property conditions.*

Each $p_n \in \mathcal{P}_n$ has a "subject/predicate/object" description[2] of the condition/requirement that is being defined (e.g., cameraResolution greaterThan 12.1MP; frequencySampling equalTo 10s). The overall population of mashup elements (from all applications) is denoted by $\mathcal{N}$.

A set of physical Things is assumed to be registered at the cloud. The owners voluntarily register/deregister physical Things to/from the cloud, meaning that CSPs must compensate the device owners for their contribution, or find some incentive mechanism for them to participate [22], [36].

Definition 2 (Physical Thing)**.** *A sensor or actuator exposing its functionality and properties as Web resources, allowing reading (e.g., temperature value) and/or update (e.g., trigger an actuation) by others. The model of a physical Thing t includes a functionality, denoted by f_t, and all properties necessary to describe it, denoted by $\mathcal{P}_t$.*

Each property $p_t \in \mathcal{P}_t$ has a "subject/predicate/object" description associated with it (e.g., cameraResolution hasValue 12.1MP). The set of all physical Things is denoted by $\mathcal{T}^{\mathrm{P}}$, while $\mathcal{P}$ and $\mathcal{F}$ are used to denote the set of all properties (e.g., sensing range, communication facility, energy consumption, location) and functionalities (e.g., image sensor), respectively.

In SAaaS, mashup elements should not be directly binded to physical Things. Instead, virtual Things should be used as intermediate entities. More specifically, each mashup element $n \in \mathcal{N}$ should be binded to a single virtual Thing, while a virtual Thing can be binded to multiple mashup elements (with same functionality and compatible property requirements). Virtual Things are then "materialized" onto physical Things.

Definition 3 (Virtual Thing)**.** *Entity built at the cloud to act on behalf of a set of compatible mashup elements. The materialization of a virtual Thing j must fulfill the requirements of all its mashup elements.*

The use of virtual Things, each requiring some virtual workspace, allows data to be consumed by multiple application mashups and allows a reduction of data

[2]A Resource Description Framework (RDF) triple. See [37].

collection/storage, increasing the usefulness of data. Resources are, therefore, better utilized. Different possible bindings between mashup elements and virtual Things will have different impacts on resource usage, cloud scalability and Quality of Experience (QoE). The set of virtual Things created in the cloud is denoted by $\mathcal{T}^{\mathrm{V}}$.

The use of semantic tools allows the cloud to find different ways of achieving a functionality. That is, a functionality $f \in \mathcal{F}$ can also be achieved by joining functionalities at multiple devices. Thus, there will be multiple ways of achieving a functionality, and each of them can be materialized in one or more devices. The set of possible materializations for functionality f is denoted by $\mathcal{M}(f)$, and $\mathcal{M}_i^f \in \mathcal{M}(f)$ denotes the i^{th} possible materialization that may include one or more devices. That is[3], $\cup_{t \in \mathcal{M}_i^f} f_t \triangleq f$, $\forall \mathcal{M}_i^f \in \mathcal{M}(f)$. A virtual Thing will be materialized using one of these materialization possibilities. At last, the SAaaS materialization problem is defined as follows.

Definition 4 (SAaaS Materialization (SASM) Problem)**.** *Given a set of applications, each with a set of components binded to Thing mashups at the cloud, assign mashup elements to virtual Things, and materialize virtual Things onto physical Things, so that the overall cost is minimized while meeting the functionality and property needs of mashup elements.*

In order to define what is meant by cost, let us assume $\eta = \{\mathcal{N}_1, \mathcal{N}_2, ..., \mathcal{N}_{|\mathcal{T}^{\mathrm{V}}|}\}$ as a feasible partition of mashup elements (all elements in each $\mathcal{N}_j$ have the same functionality requirement and compatible property requirements). That is, each $\mathcal{N}_j$ will give rise to a virtual Thing. For such partition of mashup elements, the cost of assigning materialization $\mathcal{M}_i^f$ to $\mathcal{N}_j$ will be:

$$Cost(\mathcal{M}_i^f, \mathcal{N}_j) = Gap(\mathcal{M}_i^f, \mathcal{N}_j) + Flow(\mathcal{N}_j). \quad (1)$$

The first component is the property gap cost when assigning materialization $\mathcal{M}_i^f$ to $\mathcal{N}_j$, $\mathcal{M}_i^f \in \mathcal{M}(f)$, while the second component is the flow cost or number of flows to other virtual Thing workspaces at the cloud, which is dependent on individual Thing mashup workflows. These should be normalized, and weights can be given to the two components. The property gap cost is given by:

$$Gap(\mathcal{M}_i^f, \mathcal{N}_j) = \sum_{\{p \in \chi\}} min_{n \in \mathcal{N}_j}\{\Delta_{f,i}^{n,p}\}, \quad (2)$$

where $\chi = \cup_{n' \in \mathcal{N}_j} \mathcal{P}_{n'}$ includes the set of all properties used in $\mathcal{N}_j$'s mashup element conditions, and $\Delta_{f,i}^{n,p}$ is the highest gap between the condition specified for property p in mashup element n, and p values offered by physical Things in $\mathcal{M}_i^f$. This highest gap should be captured because more than one physical Thing in $\mathcal{M}_i^f$ can have a given property. Since multiple mashup elements in $\mathcal{N}_j$ may impose conditions for property p, the *min* is used to capture the lowest $\Delta_{f,i}^{n,p}$ associated with such mashup elements. Such lowest request-supply gap, from all n's in $\mathcal{N}_j$, refers to the closest matching. As an example, let us assume two mashup elements in $\mathcal{N}_j$ requesting for different camera resolutions. If a feasible camera is supplied to both of them, the most demanding request (the one requesting for highest resolution) will be the one closer to the supplied camera resolution, having the lowest gap. The other request can be considered fulfilled.

[3]The symbol $\triangleq$ means equal by definition, in our case logically/semantically equivalent.

B. Mathematical Formulation of the SASM Problem

The following client-related information is assumed to be known:

$\boldsymbol{\mathcal{A}}$	Set of applications, where $\mathcal{A}_i \in \mathcal{A}$ refers to a specific application.
$\boldsymbol{\mathcal{C}(\mathcal{A}_i)}$	Set of independent application components at $\mathcal{A}_i \in \mathcal{A}$, where $\mathcal{C}_j^i \in \mathcal{C}(\mathcal{A}_i)$ refers to a specific component. A component is binded to a Thing mashup in the cloud, each mashup having a set of elements.
$\boldsymbol{\mathcal{N}}$	Set of all mashup elements, from all application components. That is, $\mathcal{N} = \cup_{\mathcal{A}_i \in \mathcal{A}} \mathcal{C}(\mathcal{A}_i)$.
$\boldsymbol{f_n}$	Functionality required by mashup element $n \in \mathcal{N}$.
$\boldsymbol{\mathcal{P}_n}$	Set of all property conditions of mashup element $n \in \mathcal{N}$.
$\boldsymbol{\Phi^{n,n'}}$	One if $n, n' \in \mathcal{N}$ have different functionalities or some property that makes then incompatible for materialization onto the same physical Thing; zero otherwise.
$\boldsymbol{\Omega^{n,n'}}$	One if there is a mashup flow $n \rightarrow n'$, $n, n' \in \mathcal{N}$; zero otherwise.

That is, each $n \in \mathcal{N}$ requires a functionality and imposes several property constraints, which imposes limitations on the set of mashup elements binded a virtual Thing and, consequently, its materialization onto a physical Thing. Some n may have no functionality and property requirements if used for aggregation of flows with Web services. Regarding physical Things, the known information is the following:

$\boldsymbol{\mathcal{T}^{\mathrm{P}}}$	Set of all physical Things registered at the cloud, where $t \in \mathcal{T}^{\mathrm{P}}$ is used to refer to a specific physical Thing.
$\boldsymbol{f_t}$	Functionality of physical Thing $t \in \mathcal{T}^{\mathrm{P}}$.
$\boldsymbol{\mathcal{P}_t}$	Set of properties of physical Thing $t \in \mathcal{T}^{\mathrm{P}}$.

- $\boldsymbol{\mathcal{M}(f)}$ Set of possible materializations for functionality f, where $\mathcal{M}_i^f \in \mathcal{M}(f)$ denotes the i^{th} possible materialization, which may include one or more physical Things.
- $\boldsymbol{\Delta_{f,i}^{n,p}}$ Highest gap value, from all physical Things enrolled in materialization $\mathcal{M}_i^f$, for a particular p of $n \in \mathcal{N}$.
- $\boldsymbol{\Delta^{\max}}$ Highest possible property gap.

Note that $\Phi^{n,n'}$ and $\Delta_{f,i}^{n,p}$ can be extracted using SPARQL[4] because both properties/functionality requirements and properties/functionality of physical Things are semantic-based.

The variables required to formulate the SASM problem are:

- $\boldsymbol{\alpha_i^f}$ One if the i^{th} possible materialization for functionality f, $\mathcal{M}_i^f \in \mathcal{M}(f)$, is in use; zero otherwise. The cloud will have an active virtual Thing for each $\mathcal{M}_i^f$ in use.
- $\boldsymbol{\kappa_{f,i}^t}$ One if physical Thing $t \in \mathcal{T}^{\text{P}}$ is enrolled in the materialization of virtual Thing $\mathcal{M}_i^f$; zero otherwise.
- $\boldsymbol{\beta_{f,i}^n}$ One if mashup element $n \in \mathcal{N}$ is binded to virtual Thing $\mathcal{M}_i^f$; zero otherwise.
- $\boldsymbol{\zeta_{f,i}^p}$ Highest property p gap, from all mashup elements binded to virtual Thing $\mathcal{M}_i^f$.
- $\boldsymbol{\rho_{f',i'}^{f,i}}$ One if there is flow from virtual Thing $\mathcal{M}_i^f$ to virtual Thing $\mathcal{M}_{i'}^{f'}$, zero otherwise.
- $\boldsymbol{\Upsilon}$ Total property gap cost.
- $\boldsymbol{\Psi}$ Total number of flows.

The SASM problem is mathematically formulated as follows:

– Objective function:

$$max \sum_{\{f\in\mathcal{F}\}} \sum_{\{\mathcal{M}_i^f\in\mathcal{M}(f)\}} \sum_{\{n\in\mathcal{N}\}} \beta_{f,i}^n - \frac{\Upsilon}{P\times\Delta^{\max}\times N} - \\ - \frac{\Psi}{P\times\Delta^{\max}\times N^3} \quad (3)$$

where $P = \sum_{n\in\mathcal{N}} |\mathcal{P}_n|$, $N = |\mathcal{N}|$ and $\Delta^{\max}$ is the highest possible property gap. With this goal the number of fulfilled mashup elements is first maximized and then, as a secondary goal, the total property gap cost and total number of flows (between virtual Things inside the cloud) are minimized due to their negative sign. This goal is subject to:

– Physical Thing assignment:

$$\kappa_{f,i}^t \geq \alpha_i^f, \forall f \in \mathcal{F}, \forall \mathcal{M}_i^f \in \mathcal{M}(f), \forall t \in \mathcal{M}_i^f \quad (4)$$

$$\sum_{\{f\in\mathcal{F}\}} \sum_{\{\mathcal{M}_i^f\in\mathcal{M}(f)\}} \kappa_{f,i}^t \leq 1, \forall t \in \mathcal{T}^{\text{P}} \quad (5)$$

[4]Semantic query language. See [38].

Constraints (4) state that all physical Things enrolled in a materialization (of a virtual Thing) must be in use if the virtual Thing is active. Constraints (5) force a physical Thing not to be assigned to more than one virtual Thing.

$$\sum_{\{f\in\mathcal{F}\}} \sum_{\{\mathcal{M}_i^f\in\mathcal{M}(f)\}} \sum_{\{t\notin\mathcal{M}_i^f\}} \kappa_{f,i}^t \leq 0 \quad (6)$$

Constraints (6) state that a physical Thing can not be assigned to a virtual Thing if it is not participating in the materialization (device orchestration) of such virtual Thing.

– Fulfilling functionality of mashup elements:

$$\sum_{\{\mathcal{M}_i^f\in\mathcal{M}(f)\}} \beta_{f,i}^n \leq 1, \forall n \in \mathcal{N}, f = f_n \quad (7)$$

$$\alpha_i^f \geq \beta_{f,i}^n, \forall n \in \mathcal{N}, f = f_n, \forall \mathcal{M}_i^f \in \mathcal{M}(f) \quad (8)$$

Constraints (7) ensure that the functionality required by n is fulfilled by no more than one virtual Thing. In constraints (8), virtual Things become active if at least one mashup element is binded to it.

– Fulfilling property conditions of mashup elements:

$$\beta_{f,i}^n + \beta_{f,i}^{n'} \leq 2 - \Phi^{n,n'}, \forall n, n' \in \mathcal{N}, f = f_n, \\ , \forall \mathcal{M}_i^f \in \mathcal{M}(f), \quad (9)$$

These constraints are used to ensure that two mashup elements with incompatible properties are not binded to the same virtual Thing (materialized onto the same physical Thing).

– Gap between property conditions of mashup elements and physical Thing properties:

$$\zeta_{f,i}^p \geq \beta_{f,i}^n \times \Delta_{f,i}^{n,p} - (1-\beta_{f,i}^n) \times \Delta^{\max}, \forall n \in \mathcal{N}, f = f_n, \\ , \forall \mathcal{M}_i^f \in \mathcal{M}(f), \forall p \in \mathcal{P}_n \quad (10)$$

where $\Delta_{f,i}^{n,p}$ is the highest gap value, from all physical Things in materialization $\mathcal{M}_i^f$, for a particular p of n (more than one physical Thing in $\mathcal{M}_i^f$ can have property p), and $\Delta^{\max}$ is the highest possible gap value. The $\zeta_{f,i}^p$ gives the upper bound of $\Delta_{f,i}^{n,p}$, for a set of mashup elements binded to the virtual Thing of $\mathcal{M}_i^f$. The total gap cost is given by

$$\Upsilon = \sum_{\{f\in\mathcal{F}\}} \sum_{\{\mathcal{M}_i^f\in\mathcal{M}(f)\}} \sum_{\{p\in P\}} \zeta_{f,i}^p \quad (11)$$

which is included in the objective function, for gap cost minimization.

– Flows between virtual things:

$$\rho_{f',i'}^{f,i} \geq (\beta_{f,i}^{n} + \beta_{f',i'}^{n'}) \times \Omega^{n,n'} - 1, \forall n, n' \in \mathcal{N},$$
$$, f = f_n, f' = f_{n'}, \forall \mathcal{M}_i^f \in \mathcal{M}(f), \forall \mathcal{M}_{i'}^{f'} \in \mathcal{M}(f') \quad (12)$$

where $\Omega^{n,n'}$ is given information stating if there is a mashup flow $n \rightarrow n'$. These constraints find if there is any flow between virtual Things, which depends on the bindings between mashup elements and virtual Things. The overall number of flows between virtual Things is given by

$$\Psi = \sum_{\{f \in \mathcal{F}\}} \sum_{\{\mathcal{M}_i^f \in \mathcal{M}(f)\}} \sum_{\{f' \in \mathcal{F}\}} \sum_{\{\mathcal{M}_{i'}^{f'} \in \mathcal{M}(f')\}} \rho_{f',i'}^{f,i} \quad (13)$$

which is included in the objective function, for flow minimization.

– Non-negativity assignment to variables:

$$\alpha_i^f, \kappa_{f,i}^t, \beta_{f,i}^n, \rho_{f',i'}^{f,i} \in \{0,1\}; \zeta_{f,i}^p, \Upsilon, \Psi \in \Re^+. \quad (14)$$

These constraints define the type of variables.

The CPLEX optimizer is used to solve instances of this problem. The solution found will be the optimal solution for the SASM problem instance under consideration.

C. Hardness of the Problem

Theorem 1. *The SASM problem is NP-hard.*

Proof. Considering a compatibility graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ such that $(n, n') \in \mathcal{E}$ iff $\Phi^{n,n'} = 0$, constraints (9) state that all mashup elements assigned to a virtual Thing (to be materialized onto a device) must be compatible, which corresponds to a clique subgraph in $\mathcal{G}$. When adding constraints (10)-(13), and inserting Υ and Ψ into the objective function, the clique size is to be maximized as this leads to lower Υ and Ψ values. This comes down to the maximum clique problem, which happens to be NP-hard [39]. Since multiple virtual Things are being built, the SASM problem can be seen as a multi-dimensional maximum clique problem. The SASM problem is, therefore, NP-hard. □

V. Algorithmic Approach

A. Motivation

Although optimal solutions can be obtained using the mathematical model previously presented, the hardness of the SASM problem makes it difficult to obtain solutions within a reasonable time when the instances of the problem are large. For this reason a heuristic is proposed in the following section. Its performance in then compared against the heuristic proposed in [7]. As far as known, [7] is the only work proposing a concrete solution to bind multiple Thing requests to a physical Thing, while considering Thing mashups managed in the cloud. Such approach starts with a single mashup element binded to a virtual Thing. These are randomly placed at the cloud and initialized with infinite cost. The solution is then improved as maximum cliques are extracted, from a mashup element compatibility graph $\mathcal{G}$, and assessed for their effectiveness in improving the solution.

However, and contrarily to our mathematical model, the approach in [7] gives no guarantees of finding an optimal solution, besides having some drawbacks. More specifically:

- Maximum cliques are extracted from the compatibility graph, one for each mashup element $n \in \mathcal{N}$. Finding the maximum clique is known to be NP-hard, and feasible just through heuristic techniques [39]. Among existing heuristic techniques, the greedy-based ones present great simplicity and high speed but have difficulty in finding good solutions given its myopic nature. In [7], a greedy-based approach is used.
- Only maximum cliques, from the compatibility graph $\mathcal{G}$, are considered when binding mashup elements to virtual Things. Other grouping combinations are not evaluated (subgraphs of maximum cliques), meaning that some mashup elements may not be fulfilled although there was a viable solution.
- The impact of selected virtual Thing materializations, at each step, is also not evaluated. That is, choices made in a specific step can make future virtual Thing materializations infeasible because virtual Things will compete for devices.

The heuristic proposed next has no such drawbacks because it tries to increase the size of feasible cliques around each mashup element, in parallel, while performing productive swap operations. This way different mashup element groupings are evaluated while avoiding building a myopic greedy approach. A feasible clique is one for which there is a materialization (physical Thing) available.

B. Algorithm Details

Let us assume the previously mentioned compatibility graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ such that $\{n, n'\} \in \mathcal{E}$ iff $\Phi^{n,n'} = 0$, and let $\mathcal{N}_n$ denote the neighbours of node $n \in \mathcal{N}$, $\mathcal{N}_n = \{n' \in \mathcal{N} : (n, n') \in \mathcal{E}\}$. Let us define $\mathcal{X}(\mathcal{Q})$ as the set of nodes that may expand a given clique $\mathcal{Q}$, either through direct inclusion or after a swap operation. That is, considering a given clique $\mathcal{Q} \subset \mathcal{G}$, $\mathcal{X}(\mathcal{Q}) = \{n \in \mathcal{N} : |\mathcal{Q} \backslash \mathcal{N}_n| \in \{0,1\}\}$. When $|\mathcal{Q} \backslash \mathcal{N}_n| = 0$ the clique can expand by the direct inclusion of node n, and when $|\mathcal{Q} \backslash \mathcal{N}_n| = 1$ the clique can swap one of its nodes (the one not connected to n) with n in order to diversify the attempts to expand.

As previously stated, the heuristic tries to increase the size of cliques around each mashup element, in parallel. Thus, every node will be included in a clique,

```
/* STEP: Initialization */
Q = {Q_1, Q_2, ..., Q_|N|}
i = 1
for each n ∈ N do
    Q_i ← n; i++.
end
/* STEP: Clique expansion */
repeat
    L = ∅
    for each Q_i ∈ Q : Q_i ≠ ∅ do
        Determine X(Q_i)
        for each n ∈ X(Q_i) do
            if |Q_i \ N_n| = 0 then
                L ← cost reduction scenario using Claim 1
            end
            if |Q_i \ N_n| = 1 then
                L ← cost reduction scenario using Claim 2
            end
        end
    end
    l* = BestScenario(L)
    Realize l*
until l* = ∅ ;
```

Algorithm 1: Pseudo-code of heuristic algorithm.

although with a single node at the beginning. When attempting to expand cliques, nodes may move from one clique to another and a clique can be absorbed by another. The heuristic is based on the following claims:

Claim 1. *The direct inclusion of $n \in \mathcal{X}(\mathcal{Q}_1)$ into clique $\mathcal{Q}_1$, and consequent removal from its current clique $\mathcal{Q}_2$, should only be performed if $\exists \mathcal{M}_i^f, \mathcal{M}_j^f \in \mathcal{M}(f)$ such that $Cost(\mathcal{M}_i^f, \mathcal{Q}_1 \cup \{n\}) + Cost(\mathcal{M}_j^f, \mathcal{Q}_2 \backslash \{n\}) < Cost(\mathcal{M}_i^f, \mathcal{Q}_1) + Cost(\mathcal{M}_j^f, \mathcal{Q}_2)\}$, where f is the functionality required by all nodes in $\mathcal{Q}_1$ and $\mathcal{Q}_2$. That is, there is an overall cost reduction.*

Claim 2. *The swap between $n \in \mathcal{X}(\mathcal{Q}_1)$ and $n' \in \mathcal{Q}_1$, where $(n, n') \notin \mathcal{E}$, should only be performed if $\exists \mathcal{M}_i^f, \mathcal{M}_j^f \in \mathcal{M}(f)$ such that applying Claim 1 to $\mathcal{Q}_1 \backslash \{n'\} \cup \{n\}$ compensate the cost increase associated with such swap.*

where function $Cost(,)$ has been defined in (1). For this function to be aligned with the objective function used by the mathematical model, and fair comparison between these two approaches, a small weight should be given to the second component of $Cost(,)$. Regarding Claim 2, the idea is that there will be a cost increase when swapping n with n' (current cliques were built having the best cost into account, and at the time all neighbours were evaluated), but then a direct inclusion compensates such increase. The pseudo-code of the heuristic algorithm is shown in Algorithm 1.

TABLE II
ADOPTED PARAMETER VALUES.

Parameter	Value
Functionality pool size	10
Avg size of property pools	10
Total number of devices	100
Total number of mashup elements	20-200 (x-axis)
Avg number of elements per mashup	5 or 10
Mashup element's properties (from pool)	50%
δ	0.5
$\{\Delta_1, ..., \Delta_5\}$	$\{1, ..., 5\}$

VI. Analysis of Results

A. Scenario Setup

For the evaluation of results, random scenarios were generated based on a pool of functionalities and a pool of properties, for each functionality. The physical Things and mashup elements were created as follows:

- Mashups are randomly generated using the algorithm in [40], considering an average number of elements per mashup.
- A mashup element will have its functionality requirement randomly selected from the pool of functionalities, together with a percentage of its properties.
- Any pair $n_i, n_j \in \mathcal{N}$ with the same functionality requirement, at at least with a property in common, is compatible with probability δ (incompatibility may exist due to their property conditions).
- A physical Thing has a functionality assigned to it, randomly selected from the pool of functionalities, together with all the properties associated with the selected functionality.
- The gap between a property condition and device property is randomly selected from $\{\Delta_1, ..., \Delta_5\}$, where Δ_1 is the lowest cost and Δ_5 is the highest.

Table II summarizes the adopted parameter values.

B. Evaluation

1) Fulfilled Mashup Elements and Virtual Things: Here, the number of mashup elements that have been fulfilled (mapped to a virtual Thing), the total number of generated virtual Things (successfully materialized onto devices), and the average number of mashup elements per virtual Thing (average size of cliques in compatibility subgraph $\mathcal{G}$), are analysed. Comparison will be done between the proposed mathematical optimization model (MathOptim), proposed heuristic algorithm (Heuristic), and the Highest Cost Variance (HCV) variant of the heuristic proposed in [7]. As stated in [7], the HCV variant is the one performing better, making it unnecessary to consider the other variants. These results are shown in plots of Figures 3, 4 and 5, respectively, for an increasing number of mashup elements. Regarding the mathematical model, and after obtaining the solution for instances using CPLEX, the values for the first plot are the result

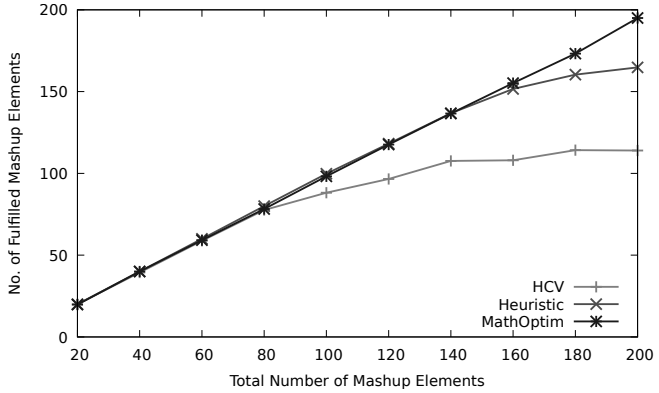

Fig. 3. Number of fulfilled mashup elements. Avg number of elements per mashup = 10.

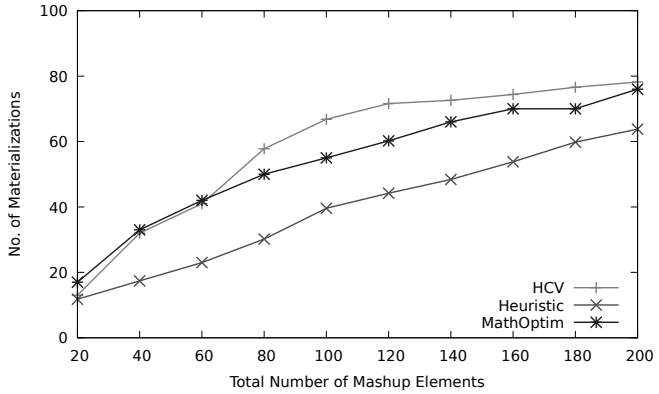

Fig. 4. Number of virtual Things materialized onto physical devices. Avg number of elements per mashup = 10.

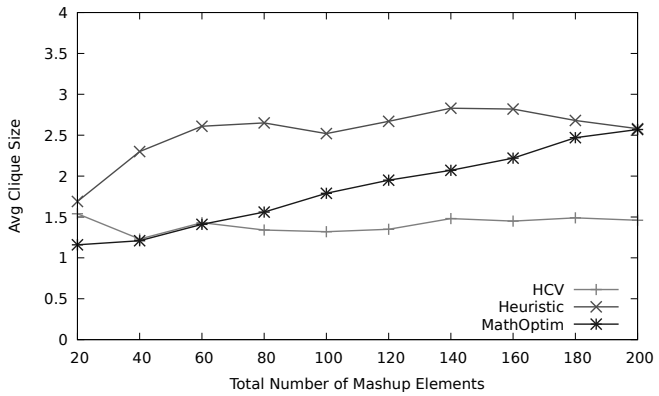

Fig. 5. Average number of fulfilled mashup elements per virtual Thing. Avg number of elements per mashup = 10.

of counting non-zero β variables, while the values for the second plot are the result of counting non-zero α variables. The last plot is built using information from the first and second plots.

From these results it is possible to observe that the new heuristic is able to fulfill more mashup elements (see plot 3) than the heuristic approach proposed in [7], closely approaching the mathematical optimization

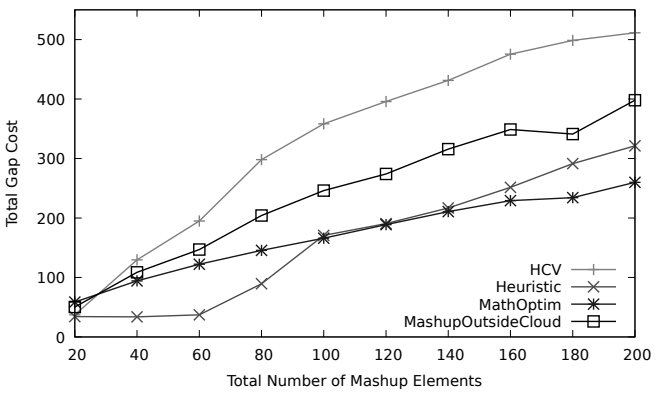

Fig. 6. Total property gap cost of materializations (Υ). Avg number of elements per mashup = 10.

model. This is more pronounced for less than 160 mashup elements. As the number of materializations (virtual Things) gets closer to the number of available devices, the proposed heuristic slows down its performance, although it still outperforms the heuristic variant from [7]. Please note that the proposed heuristic could increase the number of fulfilled elements, for such scenarios, if swap operations explore more distant neighborhoods. However, this will increase the complexity of the heuristic.

In what concerns to the number of virtual Things (materializations), the proposed heuristic is able to use less virtual Things (materializations) than all approaches, mathematical model included, meaning that more mashup elements are mapped to a virtual Thing (cliques of larger size). This makes materializations more effective and releases more devices for future materializations (see plot 5). Note that the heuristic requires less virtual Things than the mathematical optimization model because the primary goal of the mathematical model is to maximize the number of fulfilled mashup elements, and not the clique size. To maximize the clique size the mathematical model would become non-linear, making the model untreatable. However, since the materialization cost is also at the objective function (secondary component), the mathematical model has also interest in larger clique sizes. This becomes noticeable when the population of mashup elements to be fulfilled increases. In this case multiple solutions exist for the same number of fulfilled mashup elements, and then the second component can play its role. For this reason, the mathematical model keeps increasing the clique size as the number of mashup elements increases, being able to manage materializations more efficiently than the other approaches in such scenarios: provides high number of fulfilled mashup elements and large clique size.

2) Materialization Cost: Here, the total property gap cost associated with the chosen materializations,

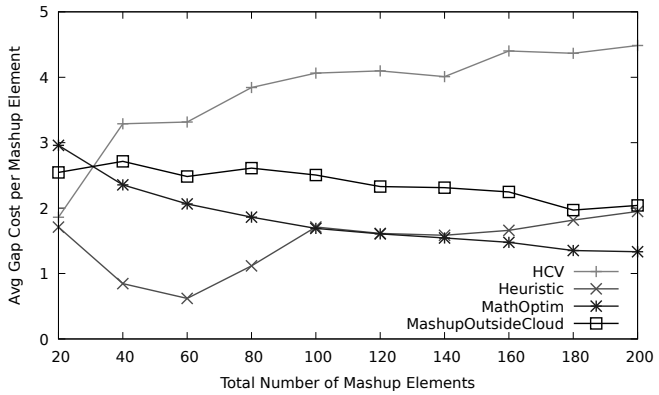

Fig. 7. Average property gap cost per mashup element. Avg number of elements per mashup = 10.

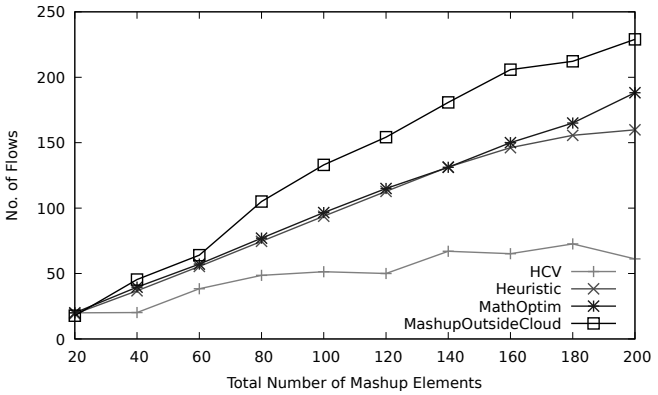

Fig. 8. Total number of flows (Ψ). Avg number of elements per mashup = 10.

and average property gap cost per mashup element, are analysed. These results are shown in plots of Figures 6 and 7, respectively, for an increasing number of mashup elements. To assess the benefits of managing Thing mashups in the cloud, when compared with their management at the client side (traditional approach), and effectiveness of the proposed approach that aims to minimize both property gap and flow costs, plots also include the results of the mathematical optimization model for a different objective function: $max \sum_{\{f \in \mathcal{F}\}} \sum_{\{\mathcal{M}_i^f \in \mathcal{M}(f)\}} \sum_{\{n \in \mathcal{N}\}} \beta_{f,i}^n$. This represents the case where Thing mashups are managed at the client side (MashupOutsideCloud). In this case the devices are individually picked by the client without taking into account other requests, and for this reason Υ and Ψ are left out.

From the results it is possible to observe that the mathematical optimization model (MathOptim) is the one finding devices more close to the needs of mashup elements (lower materialization costs) releasing the other devices for future requests. Since the gap cost is included in the objective function as a secondary goal, this effect becomes more visible when the population of mashup elements increases. That is, since many solutions exist for the same value of the first component of the objective function (number of fulfilled mashup elements), the optimizer is then able to retrieve the solution having lower gap cost (second component). The proposed heuristic presents better results than HCV from [7], meaning that materializations are choosing devices closer to the needs of the mashup elements.

Results also show that managing Thing mashups outside the cloud leads to significant property gap cost increases, as no optimized assignment of mashup elements to virtual Things is performed. This is confirmed by Figure 7 where the average property gap cost per mashup element is shown. Such cost has a more pronounced reduction when Thing mashups are managed in the cloud (MathOptim), revealing that virtual Things are built more effectively. This does not happen when devices are picked at the client (MashupOutsideCloud), although some virtual Things may still act on behalf of more than one device.

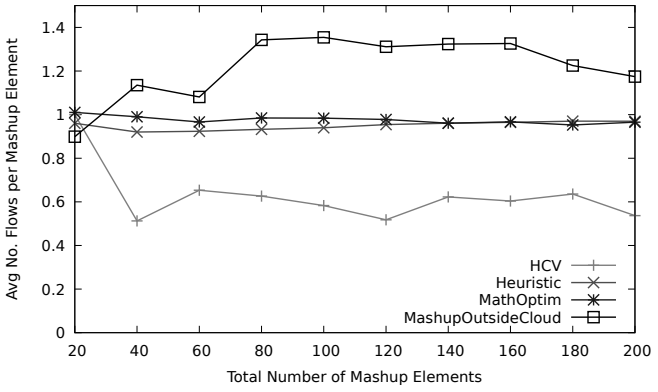

Fig. 9. Average number of flows per mashup element. Avg number of elements per mashup = 10.

3) Flows at the Cloud: This section analyses the total number of flows between virtual Thing workspaces at the cloud, and average number of flows per mashup element. These results are shown in plots of Figures 8 and 9, respectively, for an increasing number of mashup elements. The case of Thing mashups managed at the client side is also shown.

Results show that the proposed mathematical optimization model and heuristic present the best results. More specifically, although the number of flows increases, this is mainly related with the increase in the number of fulfilled mashup elements and number of virtual Things built. The average number of flows per mashup element is kept under control (see Figure 9), which means that the assignment of mashup elements to virtual Things is performed in a way that flows between virtual Thing workspaces is minimized (cliques are built taking into account mashup elements having common predecessor/successors). This is not possible when mashups are managed outside the cloud, as seen in Figure 9. The heuristic approach proposed in [7] presents a low number of flows, which

is mainly related with the low number of fulfilled mashup elements and virtual Things.

4) Size of Mashups: Figures 10 - 13 show the impact of low and high flow dependency level among mashup elements. More specifically, the average number of flows is plotted for mashups with an average size of 5 and 10 elements, meaning that the overall flow dependency among mashup elements will be lower for the first and higher for the second. Results show that the proposed mathematical optimization model (MathOptim) and heuristic perform quite well, being able to minimize flows between virtual Thing workspaces for low and high flow dependency levels. The HCV is not able to perform well when flow dependency is low, although the average number of flows per mashup element reduces slightly as the number of mashup elements increases, meaning that flow merging is embedded in its goal. This does not happen in Figure 13 because mashups are managed at the client side, meaning that it is not possible to globally optimize flows from multiple client mashups. For this reason the average number of flows per mashup element increases as the number of mashup elements increases.

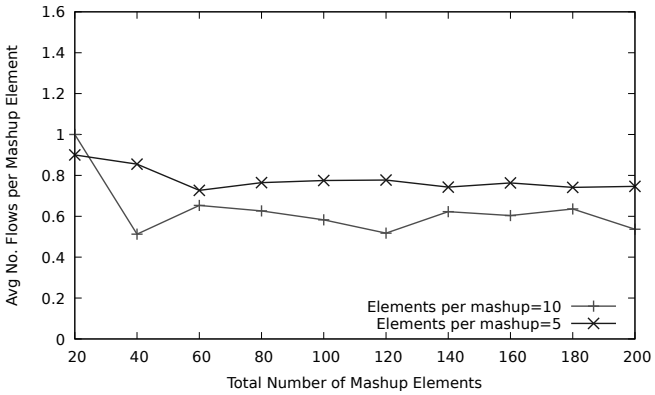

Fig. 10. Avg number of flows per mashup element for mashup sizes of 5 and 10: HCV.

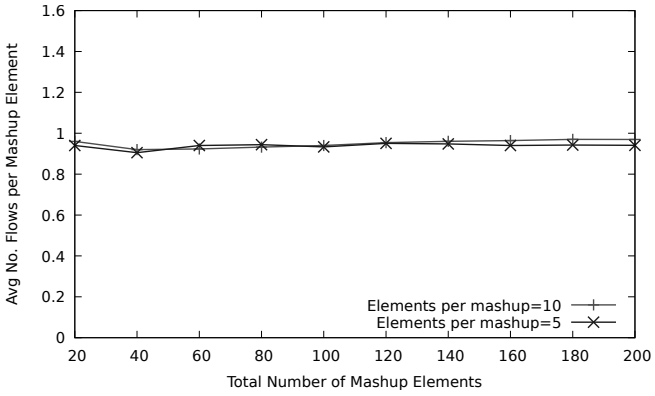

Fig. 11. Avg number of flows per mashup element for mashup sizes of 5 and 10: Heuristic.

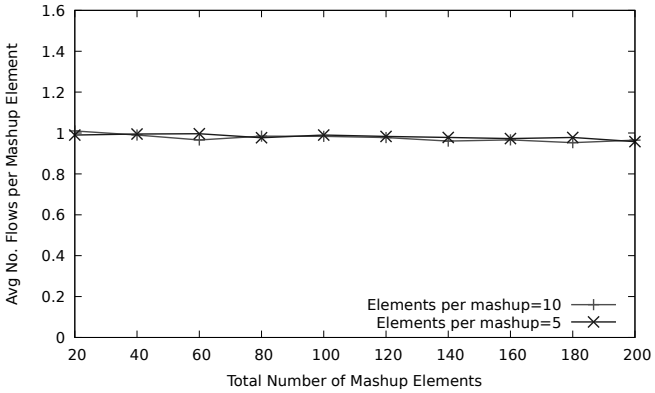

Fig. 12. Avg number of flows per mashup element for mashup sizes of 5 and 10: MathOptim.

VII. Conclusions

This article addresses resource allocation in SAaaS clouds managing virtual sensor mashups. Besides an adequate mathematical optimization model to solve this problem, a heuristic is proposed. The heuristic outperforms previous works, fulfilling more mashup elements (requests from clients) and using less physical Things for materialization of virtual Things. This means that virtual Things are acting on behalf of more mashup elements, making them more productive. The heuristic also shows lower materialization costs, which means that allocated physical Things are near to what is requested by clients, leaving physical Things with better features for future requests. Flows are also adequately optimized in order to reduce the overall data transfer between virtual workspaces of virtual Things.

Acknowledgment

This work was supported by FCT (Foundation for Science and Technology) from Portugal within CEOT (Center for Electronic, Optoelectronic and Telecommunications) and UID/MULTI/00631/2019 project.

References

[1] G. Fortino, et. al.: “Modeling Opportunistic IoT Services in Open IoT Ecosystems”, Proceedings of Workshop “From Objects to Agents”, Italy (2017).

[2] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos: “Sensing as a Service Model for Smart Cities Supported by Internet of Things”, Transactions on Emerging Telecommunications Technologies, Vol. 25, No. 1 (2014), John Wiley & Sons, Inc. New York, NY, USA.

[3] Ing-Ray Chen, et al.: “Trust-Based Service Management for MobileCloud IoT Systems”, IEEE Transactions on Network and Service Management, Vol. 16, No. 1 (2019).

[4] Dominique Guinard and Vlad Trifa: “Building the Web of Things”, Manning Publications (2016).

[5] W3C: “Web of Things (WoT) Architecture”, https://w3c.github.io/wot-architecture/ [Accessed 02-12-2019].

[6] Salvatore Distefano, Giovanni Merlino, and Antonio Puliafito: “A Utility Paradigm for IoT: The Sensing Cloud”, Pervasive and Mobile Computing, Vol. 20 (2015).

[7] Joel Guerreiro, Luís Rodrigues, and Noélia Correia: “Resource

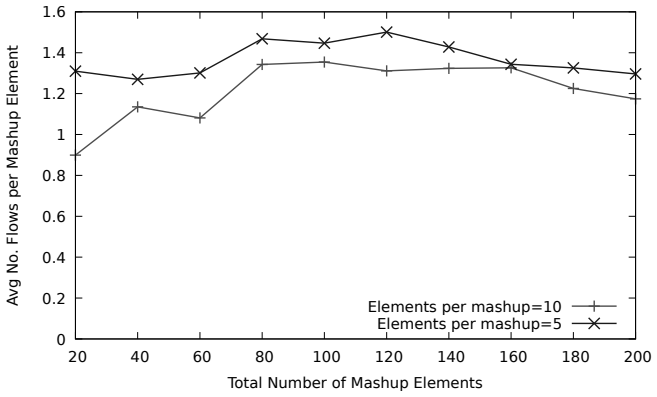

Fig. 13. Avg number of flows per mashup element for mashup sizes of 5 and 10: MashupOutsideCloud.

Allocation Model for Sensor Clouds under the Sensing as a Service Paradigm", Computers, Vol. 8, No. 1 (2019).
[8] Yucong Duan, et al.: "Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends", IEEE International Conference on Cloud Computing (2015).
[9] Shyam Patidar, Dheeraj Rane, and Pritesh Jain: "Survey Paper on Cloud Computing", International Conference on Advanced Computing & Communication Technologies (2012).
[10] Sudip Misra, Subarna Chatterjee, and Mohammad S. Obaidat: "On Theoretical Modeling of Sensor Cloud: A Paradigm Shift From Wireless Sensor Network", IEEE Systems Journal, Vol. 11, No. 2 (2017).
[11] Tian Wang, et. al.: "Sustainable and Efficient Data Collection from WSNs to Cloud", IEEE Transactions on Sustainable Computing, Vol. 4, No. 2 (2019)
[12] L.P. Dinesh Kumar, et al.: "Data Filtering in Wireless Sensor Networks using Neural Networks for Storage in Cloud", International Conference ICRTIT (2012).
[13] T. Dinh and Y. Kim: "An Efficient Sensor-Cloud Interactive Model for On-Demand Latency Requirement Guarantee" IEEE International Conference on Communications (2017).
[14] A. Deshwal, S. Kohli, and K. P. Chethan: "Information as a Service based Architectural Solution for WSN", IEEE International Conference on Communications in China (2012).
[15] Riccardo Petrolo, Valeria Loscrì and Nathalie Mitton: "Towards a Smart City Based on Cloud of Things, a Survey on the Smart City Vision and Paradigms", Transactions on Emerging Telecommunications Technologies, Wiley Online (2015).
[16] Sudip Misra, et al.: "Optimal Gateway Selection in Sensor-Cloud Framework for Health Monitoring", IET Wireless Sensor Systems, Vol. 4, No. 2 (2014).
[17] Yao-Chung Hsu, Chi-Han Lin, and Wen-Tsuen Chen: "Design of a Sensing Service Architecture for Internet of Things with Semantic Sensor Selection", International Conference UTC-ATC-ScalCom (2014).
[18] Chin-Feng Lai, Han-Chieh Chao, Ying-Xun Lai, and Jiafu Wan: "Cloud-Assisted Real-Time Transrating for HTTP Live Streaming", IEEE Wireless Communications, Vol. 20, No. 3 (2013).
[19] Chin-Feng Lai, Honggang Wang, Han-Chieh Chao, and Guofang Nan: "A Network and Device Aware QoS Approach for Cloud-Based Mobile Streaming", IEEE Transactions on Multimedia, Vol. 15, No. 4 (2013).
[20] Wei Wang, Qin Wang, and Kazem Sohraby: "Multimedia Sensing as a Service (MSaaS): Exploring Resource Saving Potentials of at Cloud-Edge IoTs and Fogs", IEEE Internet of Things Journal, Vol. 4, No. 2 (2017).
[21] Y. Xu and S. Mao: "A Survey of Mobile Cloud Computing for Rich Media Applications", IEEE Wireless Communications, Vol. 20, No. 3 (2013).
[22] Xiang Sheng, Jian Tang, Xuejie Xiao, and Guoliang Xue: "Sensing as a Service: Challenges, Solutions and Future Directions", IEEE Sensors Journal, Vol. 13, No. 10 (2013).
[23] M. A. E. Al-Fagih, F. M. Al-Turjman, W. M. Alsalih, and H. S. Hassanein: "Priced Public Sensing Framework for Heterogeneous IoT Architectures", IEEE Transactions on Emerging Topics in Computing, Vol. 1, No. 1 (2013).
[24] Chongwu Dong, et. al., "A Novel Distribution Service Policy for Crowdsourced Live Streaming in Cloud Platform" IEEE Transactions on Network and Service Management, Vol. 15, No. 2 (2018)
[25] Jaeho Kim and Jang-Won Lee: "OpenIoT: An Open Service Framework for the Internet of Things", IEEE World Forum on Internet of Things (2015)
[26] Mihui Kim, Mihir Asthana, Siddhartha Bhargava, Kartik Krishnan Iyyer, Rohan Tangadpalliwar and Jerry Gao: "Developing an On-Demand Cloud-Based Sensing-as-a-Service System for Internet of Things", Journal of Computer Networks and Communications, Vol. 2016, Article ID 3292783.
[27] E. AL-Hawri, N.Correia, and A.Barradas: "Design of Network Coding Based Reliable Sensor Networks" Ad Hoc Networks, Elsevier, Vol. 91 (2019)
[28] Francesco Longo, et.al.: "Stack4Things: an OpenStack-based Framework for IoT", International Conference on Future Internet of Things and Cloud (2015)
[29] Hyeontaek Oh, et. al.: "Web Intent based Service Mashups for IoT Platform", IEEE Global Conference on Consumer Electronics (2014)
[30] Sehyeon Heo, Sungpil Woo, Janggwan Im, and Daeyoung Kim: "IoT-MAP: IoT Mashup Application Platform for the Flexible IoT Ecosystem", International Conference on the Internet of Things (2015)
[31] Xiongnan Jin, et. al.: "Automated Mashup of CoAP Services on the Internet of Things", IEEE World Forum on Internet of Things (2015)
[32] Sungkwang Eom, Wonwoo Ro, and Kyong-Ho Lee: "A Semantic Sensor Mashup Platform for Internet of Things", IEEE World Forum on Internet of Things (2018)
[33] Janggwan Im, Seonghoon Kim, and Daeyoung Kim: "IoT Mashup as a Service: Cloud-Based Mashup Service for the Internet of Thing", IEEE International Conference on Services Computing (2013)
[34] Bo Cheng, Shuai Zhao, Junyan Qian, Zhongyi Zhai, and Junliang Chen: "Lightweight Service Mashup Middleware With REST Style Architecture for IoT Applications", IEEE Transactions on Network and Service Management, Vol. 15, No. 3 (2018).
[35] Michael Compton, et al.: "The SSN Ontology of the W3C Semantic Sensor Network Incubator Group", Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 17 (2012).
[36] Maryam Pouryazdan, et al.: "Quantifying User Reputation Scores, Data Trustworthiness, and User Incentives in Mobile Crowd-Sensing", IEEE Access, Vol. 5 (2017).
[37] W3C: "Semantic Web W3C", https://www.w3.org/standards/semanticweb/ [Accessed 02-12-2019].
[38] W3C: "SPARQL Query Language for RDF", https://www.w3.org/TR/rdf-sparql-query/ [Accessed 02-12-2019].
[39] Alexandre Prusch Züge and Renato Carmo: "On Comparing Algorithms for the Maximum Clique Problem", Discrete Applied Mathematics, Vol. 247 (2018).
[40] F.A. Onat and I. Stojmenovic: "Generating Random Graphs for Wireless Actuator Networks", IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (2007).