2018 IEEE 16th Int. Conf. on Dependable, Autonomic & Secure Comp., 16th Int. Conf. on Pervasive Intelligence & Comp., 4th Int. Conf. on Big Data Intelligence & Comp., and 3rd Cyber Sci. & Tech. Cong.

A System Architecture for Cloud of Sensors

Igor L. dos Santos
Centro Federal de Educação Tecnológica Celso Suckow da Fonseca
Rio de Janeiro, Brazil
igor.santos@cefet-rj.br

Marcelo P. Alves, Flavia C. Delicato, Paulo F. Pires, Luci Pirmez
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil
fdelicato@gmail.com
luci.pirmez@gmail.com

Wei Li, Albert Y. Zomaya
The University of Sydney
Sydney, Australia
liwei@it.usyd.edu.au,
albert.zomaya@sydney.edu.au

Samee U. Khan
North Dakota State University
North Dakota, USA
samee.khan@ndsu.edu

***Abstract*— The Cloud of Sensors (CoS) paradigm has emerged from the broader concept of Cloud of Things. CoS infrastructures have the potential to leverage the benefits of both Clouds and wireless sensor and actuator networks (WSAN). As traditional Cloud platforms, CoS are built on the virtualization principles. A major challenge in the development of CoS virtualization models is energy consumption, as sensors are well-known to be energy constrained. We claim that the design decisions for the specification of software artifacts composing a CoS must consider the energy efficiency as a primary requirement. Edge computing has recently raised as a paradigm to leverage computing and networking capabilities at the network edge thus, allowing a more decentralized design. In this paper, we investigate a new model that integrates cloud, edge, and WSANs in a CoS infrastructure and propose a novel three-tier system architecture. We describe an archetype that supports implementing a three-tier CoS system, fostering energy efficiency and sustainability. We also discuss issues on implementing the CoS as a real software system, running on top of FIWARE.**

Keywords—Cloud of Sensors, edge computing, energy, architecture.

I. INTRODUCTION

In the last years, the need to connect Wireless Sensor and Actuator Networks (WSANs) [9] to the cloud has motivated the emergence of the Cloud of Sensors (CoS) paradigm [8]. The features of Cloud computing make it a suitable choice to process and store a large amount of data generated by smart sensors. In this sense, the cloud acts as an intermediate layer between smart sensors and applications generating the typical two-tier CoS architecture, including the sensors and cloud tiers.

Despite the advantages of cloud computing, its increasing adoption is having a significant impact on the environment since the energy consumption of data centers and the resulting emissions are growing. Therefore, data center managers are seeking greener deployments that reduce energy consumption and carbon emissions. Some authors [17] claim that the emerging edge computing paradigm is a more sustainable extension of cloud computing. The edge leverages computation and networking capabilities at the edge of the network, allowing a more decentralized and greener network design and deployment [17]. Edge nodes perform several tasks, such as collecting the data from smart things and preprocessing/ filtering it, to reconstruct the data in a more useful way. The first tier of the Edge consumes locally the portion of data generated by sensors that require real-time processing (millisecond scale). The remainder data is forwarded to the higher tiers, for operations with less stringent time constraints (seconds/minutes). The higher the tier, more extensive is its geographical coverage, the longer is its time scale, and the more energy-intensive it is. The cloud provides the ultimate and global coverage, and acts as a long-term data repository, besides allowing more complex data analytics [5].

A CoS infrastructure is composed of virtual nodes (VNs), which are abstract representations of sensors, built by using a CoS virtualization model. By hiding from users the infrastructure complexity and heterogeneity, CoS virtualization models facilitate application delivery and promote resource sharing among multiple applications. Considering the energy constraints typical of sensor nodes, as well as the current search for greener and more sustainable designs of cloud data centers, a major challenge in CoS virtualization regards the energy consumption of the infrastructure.

In a previous work, our research group proposed Olympus [8], a decentralized and information fusion-based CoS virtualization model. Olympus decentralized model aims at saving energy resources from sensor nodes by avoiding the blind transmission of raw sensing data to remote data centers and instead, leveraging the in-network processing. Olympus was a theoretical model and considered a 2-tier design of CoS systems, with a lower tier composed of WSAN and a top tier encompassing the Cloud.

In this paper, we propose a novel three-tier CoS system architecture based on Olympus [8]. In our novel architecture, we included the edge tier in the original two-tier infrastructure of Olympus. Moreover, we concretize the original concepts behind Olympus into a set of CoS system components that allow the execution of each function of Olympus' CoS virtualization.

Our main contribution is a set of guidelines, in the form of a unified archetype, targeting researchers, analysts, and practitioners from industry and academia that are interested in implementing the three-tier CoS system. As a second

978-1-5386-7518-2/18/$31.00 ©2018 IEEE
DOI 10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00118

IEEE computer society

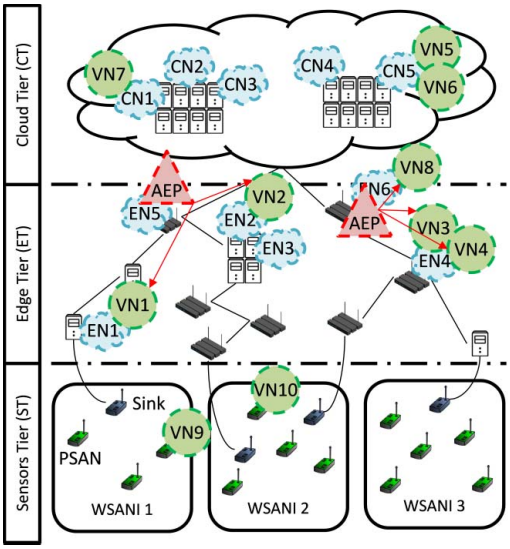

Fig. 1. The three-tier CoS infrastructure, serving applications through an overlay network of VNs

contribution, by decentralizing the CoS virtualization functions and leveraging the edge tier, our proposal follows a green network design. Finally, we discuss how to implement an instance of our architecture as a real software system, using the FIWARE platform [6]. FIWARE provides basic building blocks for Future Internet services, and follows three-tier architecture, encompassing the edge tier.

II. The Three-Tier CoS Infrastructure

We consider a three-tier CoS infrastructure (Fig. 1) composed of the sensors tier (ST), edge tier (ET) and cloud tier (CT). The ST encompasses physical sensor and actuator nodes (PSANs) deployed over a geographical area and connected by wireless links, forming Wireless Sensor and Actuator Networks infrastructures (WSANIs). Each WSANI is owned and administered by an Infrastructure Provider (InP), and every PSAN pertains to a single WSANI. The InPs define the physical and administrative boundaries of WSANIs and choose the under-lying protocols [16] used to form the paths used by PSANs to reach the WSANI's sink node. Each PSAN has the following properties: processing speed, total memory, bandwidth, sensing and actuation units, remaining energy, WSANI identification and geographical location. The ET is the middle tier, and it encompasses a set of Edge devices placed geographically closer to the Sensor tier, i.e., they are closest to the data sources.

The edge and the cloud typically provide virtualized resources. Thus, we define the virtual entities edge nodes (ENs) and cloud nodes (CNs), hosted by the physical devices of the ET and CT, respectively. We describe each EN and CN in terms of the same properties: processing speed, total memory, bandwidth and geographical location. The ENs and CNs' deployment on their respective physical hosts is transparent to the CoS virtualization model and should be handled by typical cloud and edge virtualization models, whose properties we do not discuss in detail. We follow the principle of overlay virtualization [9], respective to build VNs in an overlay layer, over either PSANs or readily available ENs and CNs. By being distributed, lacking central control and allowing resource sharing, overlays are ideal candidates for CoS virtualization [9]. In this overlay, applications to arrive in the CoS system by application entry points (AEP), to be served by VNs. The CoS virtualization model defines how to provide to applications the capabilities of PSANs, ENs, and CNs through VNs.

This work was partially supported by Brazilian Funding Agencies FAPERJ (under grant E-26/202.807/2015 for Flavia C. Delicato) and CNPq (under grants 307378/2014-4 for Flavia C. Delicato and 311508/2017-0 for Paulo F. Pires). Flavia Delicato, Luci Pirmez and Paulo Pires are CNPq fellows.

A. CoS Virtualization

In this Section, we review the main features of Olympus [8] [7]. We describe applications, VNs and the CoS virtualization functions, including resource provisioning, resource allocation and task scheduling.

We consider an application as an abstract workflow [2], describing interactions among the data provided by VNs. Each activity of the workflow is an application request, whose main goal is to demand data from a VN. We call this data the output data of a request. To meet a request, a VN may require as input the output data from a predecessor request. In this sense, the requests of an application form a directed acyclic graph (DAG) [7], whose edges model data dependencies among requests. We consider only time-based applications [7]. They define the exact time for demanding data from a VN, thus complying with a pull model, where the VN forwards data only when defined by the application. Additionally, a request defines non-negotiable (must be integrally fulfilled) and negotiable (allow partial fulfillment) requirements, which the VN must fulfill when meeting the request. Among the non-negotiable requirements, the data type is a key attribute. Each data type is unique and can be provided by a variable number of VNs. The Infrastructure Providers (InPs) define and describe the data types available in the CoS. Data types may refer to simple types of sensing data (as temperature) or to more complex data (as aggregate data or events). Among the negotiable requirements, data freshness is a key one [12]. The relevance of data freshness increases particularly since the CoS comprises a set of autonomous data sources, where data integration with different freshness values of data freshness can lead to semantic problems, hindering the execution of applications. Among several definitions, we consider data freshness as the time elapsed since data acquisition.

The CoS comprises several VNs that simplify the representation of the infrastructure to users, abstracting the underlying nodes composing the CoS. A VN is responsible for updating its provided data. Therefore, each VN coordinates its underlying infrastructure to perform tasks such as sensing, processing or actuation. The VN program, which implements the abstract representation of smart sensors, coordinates this engagement using a typical WSAN task scheduling and execution algorithm [7]. It also

implements the information fusion technique (possibly along with task scheduling and execution) to process the data obtained from PSANs and generate the output data of the given data type provided by the VN. The entire VN program's procedure, called a data update, is transparent to users/applications. The VN can avoid frequent data updates to meet future requests by simply re-using data from previous data updates that meet the request's data freshness requirement. This approach has a direct effect on the CoS system overhead (thus, on energy consumption), being essential for a greener network design. The decision on whether performing a data update or not, is part of the CoS resource allocation process. Each VN exposes a data provisioning service, whose completion consists of performing a resource allocation decision received as input. This decision is obtained by solving the typically NP-complete problem [7] of resource allocation in CoS. A resource allocation algorithm is required to make decisions, which may be either centralized or decentralized [7]. Centralized algorithms, as those based on linear programming techniques, run within an entity of the CoS (a CN, for instance) that centralizes decisions for a set of VNs. Decentralized algorithms, such as game theory-based algorithms, run within each of the VNs, which will cooperatively make resource allocation decisions.

Finally, the resource provisioning process [4] is responsible for provisioning computational resources and to instantiate VNs. Ideally, resource provisioning should be reactive, where a dynamic (real-time), elastic, transparent and automated algorithm instantiates the VN, in response to an application request. However, a proactive approach can be supported simultaneously, where InPs proactively instantiate the VN, at a time prior to the need of its allocation to an application request. VNs not being used can go into sleep mode, temporary releasing their underlying infrastructure.

III. Applying the concepts behind Olympus

In this Section, we present the architecture [1] proposed to implement the concepts behind Olympus [8]. Our architecture comprises four subsystems: CoS Manager Subsystem, Application Manager Subsystem, Virtualization Subsystem and the Virtual Node Subsystem.

The CoS Manager Subsystem provides an API that allows infrastructure providers (InPs) to manage the CoS system. This API allows describing and storing the data types handled by the CoS, along with dependencies among types and information fusion techniques available. The API also allows provisioning of VN instances proactively (as well as deleting and reconfiguring VNs), according to InPs' criteria.

The Application Manager Subsystem provides an API that allows users to express their requests for the data types available in the CoS, stored in the CoS Manager Subsystem. To discover and select the available data types, we suggest using typical service discovery methods [3]. To express and manage application workflows, we suggest using domain specific languages [15]. Users can submit their workflows through this API to be performed by the CoS.

The Virtualization Subsystem comprises six components to support VN management: Virtualization Subsystem Manager, VN Instances Repository, Registries Repository, Centralized Resource Allocation Manager, Centralized Resource Provisioning Manager and Application Entry Point.

The Virtualization Subsystem Manager coordinates the actions of the other Virtualization Subsystem components. So, when the Virtualization Subsystem boots, the Virtualization Subsystem Manager starts these components. The Application Entry Point handles the arrivals of applications in the CoS, immediately transmitting them to the Centralized Resource Allocation Manager (CRAM). The CRAM and Centralized Resource Provisioning Manager (CRPM) perform the centralized parts of, respectively, the resource allocation and resource provisioning algorithms. The CRAM is responsible for assigning VNs to application requests. The CRPM handles the instantiation and management of VNs. The Virtualization Subsystem also comprises the Registries Repository, to store the registries and locations of PSANs and VNs existing in the CoS. Registries Repository can be implemented either in a centralized manner in a CN or distributed among several CNs and/or ENs. In the latter case, the Registries Repository of each EN comprises registries of VNs and PSANs that are within the scope of the EN local region. Such approach takes advantage of the location-awareness capability of the edge tier, and it is more energy efficient. In the former case, the CN comprises a broader area (global), possibly comprising registries of all existing VNs and PSANs. Upon receiving a registry from either a VN or a PSAN, the Registries Repository (RR) decides about disseminating this registry among other nodes implementing the RR. The VN Instances Repository (VNIR) stores various VN instances locally, thus, several VNs run within each VNIR.

The Virtual Node Subsystem comprises the VN Manager, Decentralized Resource Provisioning Manager, Decentralized Resource Allocation Manager, Data Provisioning Manager, Data Storage Manager, and Sensing and Actuation Manager.

The VN Manager is the first component of the Virtual Node Subsystem to boot, and it coordinates the execution of the other components of a VN. It also handles the data communication among VNs, forming a virtual network [9], so that VNs can exchange data to accomplish the application workflows. The Decentralized Resource Allocation Manager (DRAM) and Decentralized Resource Provisioning Manager (DRPM) perform the decentralized parts of the resource allocation and resource provisioning. The DRPM boots along with the VN Manager, and performs the initialization of the VN, storing its configurations. It is also responsible to proactively publish and keep updated VN's information in the local Registries Repository. The Data Provisioning Manager (DPM) provides data in response to application requests, also providing values to the negotiable and non-

negotiable application requirements, to fulfill them. In addition, the DPM provides information regarding the description of its data provisioning service. We suggest using a well-established standard [13] for describing such information. DPM implements the data provisioning service, and performs a resource allocation decision, accessing either the Data Storage Manager (DSM) or Sensing and Actuation Manager (SAM) for service completion. The DSM manages the local memory of the VN by providing functions for accessing this local memory and inserting/retrieving historical data from it. The SAM implements the functions of the VN program for performing data updates, as aforementioned, and store the recently updated data directly into the DSM.

A. Key design decisions for CoS implementation

There are two key decisions to define the CoS operation, implementation and consequently its energy consumption. The first one concerns the type of resource allocation technique used by the CoS. Centralized techniques are usually more accurate in decision-making (achieving optimal solutions) than decentralized ones. However, they tend to impose a higher communication/energy overhead on the system, since the central decision-making node must obtain information from all the subordinated VNs. As an attempt to balance this trade-off as best as possible, ensuring a greener CoS system design, we suggest using a hybrid algorithm. Hybrid algorithms comprise centralized and decentralized decision phases. By leveraging the edge tier, it is possible to perform the centralized decision phase within an EN, for a localized vicinity of VNs under its scope. Meanwhile, the decentralized decision is performed within VNs. Moreover, to further reduce the computation time (and energy) used to make decisions in both phases, we suggest using heuristic approaches. They are known for being able to achieve near-optimal solutions in reduced computation time for NP-complete problems.

The second decision regards choosing the tier to host and run VNs, and as such, we chose the ET. ENs have greater computation and communication capabilities than PSANs, to better manage VNs. Moreover, ENs are in a privileged position, in relation to the CNs, for linking PSANs from different WSANIs under the same VN. Finally, hosting VNs in the same tier of the decision-making ENs shortens the distances between such nodes, saving energy for the CoS by reducing the communication overhead. Therefore, only ENs will host the Virtualization Subsystem and the VNs. CoS Manager and Application Manager Subsystems can fully run in the CT. They may take advantage of centralized data storages in the cloud to perform their functions. Finally, an application may arrive at an Application Entry Point either through the Internet or from a user device (such as a PC or smartphone) connected directly to an EN.

IV. System Operation

Users submit their application workflows to the CoS. The Application Manager Subsystem communicates with Application Entry Points (AEPs) which are close to VNs (or PSANs) that provide the data types required by the application. We assume the adoption of an algorithm to select the best AEP according to criteria chosen by Infrastructure Providers. The specification of this algorithm is out of the scope of this work, but an example is [11]. The workflow arrives in the selected AEP, which redirects the application to the Centralized Resource Allocation Manager (CRAM) at the same EN. The CRAM makes decisions periodically over a buffered set of applications. When the current buffering period expires, the centralized phase of the resource allocation technique occurs. A new VN may need to be instantiated to serve some application requests. In this case, the CRAM calls the Centralized Resource Provisioning Manager (CRPM), within the same EN, to perform the centralized part of resource provisioning. The CRPM calls the VN Instances Repository (VNIR) (either local or at another EN), to instantiate the VN. The VNIR starts the VN Manager of the respective VN, along with the Decentralized Resource Provisioning Manager (DRPM). The VNIR returns the control to the CRPM, which calls the DRPM. The DRPM may perform a decentralized part of resource provisioning and start the other VN's components. Finally, the DRPM registers the VN instance, calling the Registries Repository locally. Then, it returns the control to the CRAM.

When all requests are assigned to a VN, the CRAM calls the Decentralized Resource Allocation Manager (DRAM) of each selected VN. If no VN is available to meet a request (and no new VNs can be created), the application is refused and returned to the original AEP. Upon arriving at the DRAM of the respective VN, the request is queued, waiting to be served serially by the Data Provisioning Manager (DPM). The request also waits for the arrival of any input data (through the VN Manager), to free its precedence restrictions. When the request is removed from queue, the DPM is called. According to the decisions made by the CRAM/DRAM (forwarded along with the request), the DPM may perform a data update or return historical data. In the first case, it accesses the Sensing and Actuation Manager, which will perform the data update and store the fresh data into the Data Storage Manager (DSM). In the second case, the DPM will access the DSM directly, to retrieve the most recent data. Then, the request is met by the DPM. If this request has a successor request in its application workflow, its data is transmitted to the VN allocated to the successor request. Otherwise, this is the last workflow's request, so its data is transmitted back to the original AEP. The VN Manager is called by the DRAM to perform such transmissions. Then, the final data is delivered to the application user.

V. Implementation of CoS in FIWARE

The FIWARE project [6] provides an ecosystem that facilitates the development of smart applications in multiple

domains. FIWARE is organized in seven technical chapters: (i) Cloud Hosting, (ii) Data/Context Management, (iii) IoT Services Enablement, (iv) Security, (v) Applications, Services and Data Delivery, (vi) Interface to Networks and Devices Architecture, and (vii) Advanced Web-based User Interface. To the best of our knowledge, FIWARE is the first initiative that provides all the features required to implement our proposed architecture. FIWARE allows integrating smart things to the cloud. Moreover, it introduces an innovative infrastructure composed of public, reusable and generic building blocks, software components called Generic Enablers (GEs). Each GE provides well-defined APIs and interoperable interfaces that comply with the specifications published for that GE, easing application development. FIWARE is based on lightweight virtualization, using Docker containers to run GEs [6].

In our implementation [10] (Fig. 2), the FIWARE GEs described in chapters (ii) and (iii) encompass our cloud and edge tiers. However, some GEs are originally implemented centralized at the cloud. To meet the requirements of our architecture, we modified some GEs to execute in a decentralized way. Chapter (iii) provides GEs allowing the interaction of FIWARE-based applications with real-world objects. These GEs are separated in IoT Backend and IoT Edge domains. IoT Backend comprises the GEs hosted in the cloud that provide functionalities such as device managing, device composition, and discovery. IoT Edge encompasses infrastructure elements required to connect physical devices

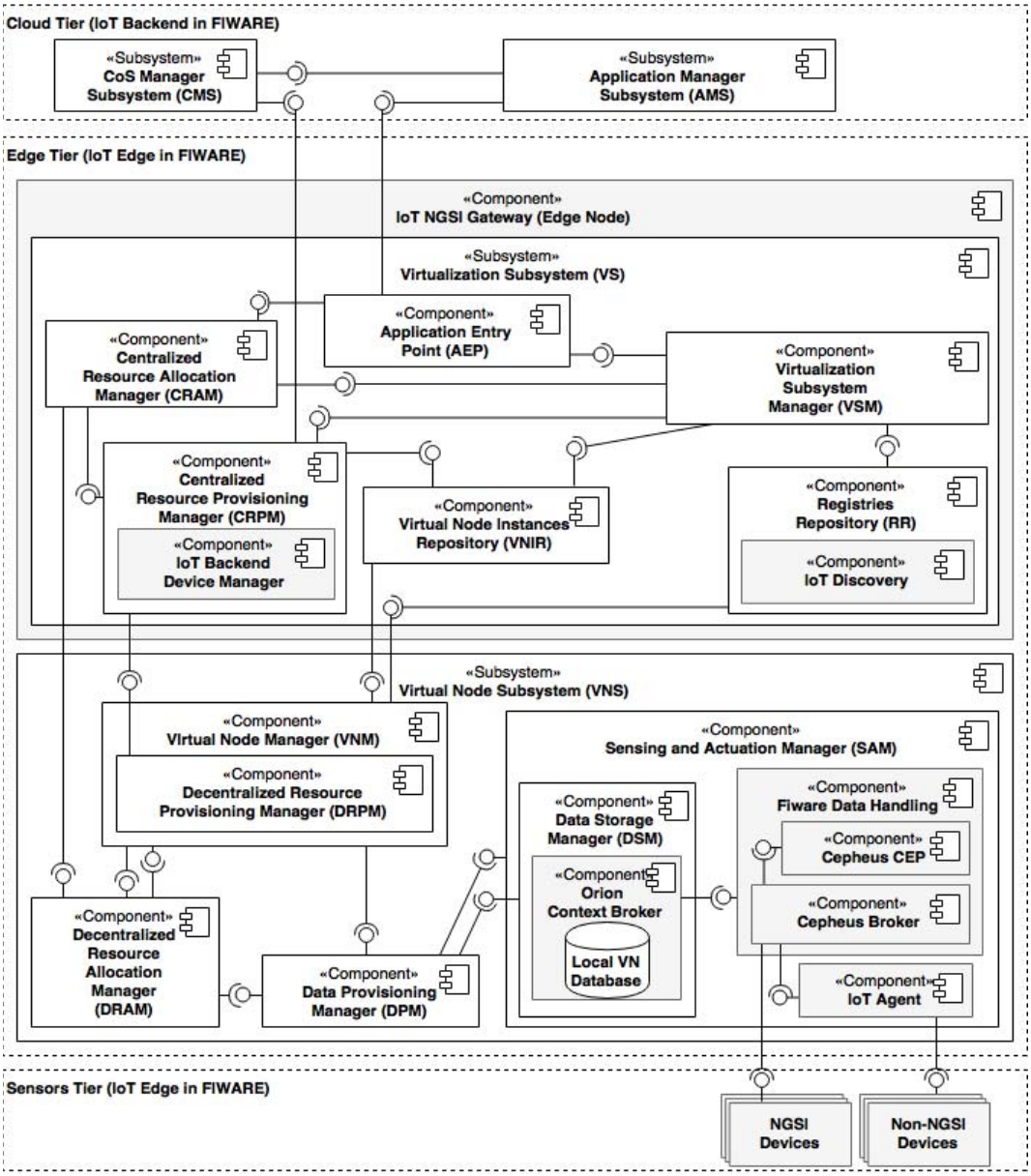

Fig. 2. Implementing CoS systems using FIWARE GEs

to FIWARE applications. It is worth mentioning that we do not implement a cloud. We only depict the typical CoS services that run on the cloud tier. Hence, we discarded chapter (i), whose components are designed to implement a cloud platform.

Table I [14] describes the GEs that supply the storage and communication infrastructure of the CoS. Our cloud tier is directly related to FIWARE's IoT Backend tier, while FIWARE's IoT Edge is related to our edge and sensor tiers. In the sensors tier, PSANs can implement FIWARE's NGSI9/10 protocol, or a conversion can be performed by the IoT Agent GE (based on HTTP/MQTT), running within a VN. For communicating directly with FIWARE's GEs, all the components in our architecture use interfaces that comply with FIWARE's NGSI9/10. In addition, the NGSI Gateway GE implements our EN, and the Virtualization Subsystem runs on top of it.

In the Virtual Node Subsystem, the Sensing and Actuation Manager (SAM) uses the Data Storage Manager (DSM), the Data Handling (DHge) and IoT Agent GEs. The DSM uses the Orion Context Broker GE (CBge), which provides context information (e.g., room temperature, citizen name/surname, a bus location) to the Data Provisioning Manager, besides performing local storage within a VN using a Mongo database [6]. Besides broker functions, the DSM has an internal structure for implementing the data model used to store historical data, and database queries to retrieve data. The implementation of task scheduling and execution is specific, and thus performed by the SAM with no direct mapping in FIWARE. The DHge, composed of Cepheus Broker and Cepheus CEP, is used to implement information fusion procedures. Cepheus Broker is a lightweight broker used for communication between devices and the DSM. The Cepheus CEP is used to process queries

TABLE I. MAPPING BETWEEN COS COMPONENTS AND FIWARE GES

CoS Elements	FIWARE GE	Description
PSANs	NGSI9/10, IoT Agent	Communication between the ST and ET. FIWARE NGSI is a RESTFul/HTTP API. NGSI9 is used to exchange information about the availability of context information whereas the purpose of NGSI10 is to exchange context information. The IoT Agent is the software module handling IoT Specific protocols.
Sensing and actuator Manager	Data Handling	Addresses the need for filtering, aggregating and merging real-time data from different sources. It includes a Broker and a CEP component.
Data Storage Manager	Orion Context Broker	Publish/Subscribe Broker, which allows applications to interchange heterogeneous events following a standard pub– sub paradigm.
Registries Repository	IoT Discovery	A service discovery mechanism that allows Context Producers (e.g., IoT Agents and Data Handling) to register sensors, actuators, and Things. Also, it enables Context consumers to discover the former registered elements.
Centralized Resource Provisioning Manager	IoT Backend Device Management	Enables creation and configuration of IoT Agents that connect to sensor networks.
-	IoT Broker	It is an IoT Backend enabler that serves as a middleware which enables fast and easy access to information about devices and their attributes.
Application Manager Subsystem	-	API for users to request sensing data
CoS Manager Subsystem	-	API to manage the CoS system
Decentralized Resource Allocation Manager & Centralized Resource Allocation Manager	-	Execute the orchestration and distribution of data.
Decentralized Resource Provisioning Manager	-	Carries out the decentralized parts of resource provisioning algorithms besides executing the VN boot.

on PSANs' data and generate higher-level aggregated events defined by the InPs. The Decentralized Resource Allocation Manager (DRAM)/Centralized Resource Allocation Manager (CRAM) perform the orchestration and distribution of data provided by the DHge, to meet application requests [6], similarly to the IoT Broker's goal. The DRAM/CRAM performs part of these functions using a hybrid resource allocation technique, not implemented by any GE. The Centralized Resource Provisioning Manager (CRPM) and IoT Backend Device Management GE (BDMge) have the same goals: enabling creation/configuration of IoT Agents that connect to PSANs. The CRPM can use the BDMge in part since it concentrates the APIs to manage PSANs. However, it is necessary to move the BDMge to run on the edge tier. Similarly, the Decentralized Resource Provisioning Manager (DRPM) can use part of the BDMge in its implementation, but the GE should be decentralized. Moreover, the DRPM is in charge of booting the VN, thus it should be implemented based on docker-compose scripts. The Registries Repository can be implemented using the IoT Discovery GE. However, this GE originally runs centralized in the cloud. It is necessary to decentralize its procedures to run at the edge.

Currently, there is no API in FIWARE to perform CoS Manager Subsystem' functions, such as creating the data model used within VNs to export data or creating SQLs to perform data fusion within the CEP environment. In the first case, the activity is performed by the InP, using JSON. The created model is then provided to the Context Broker using an HTTP POST. In the second case, an interface provided by the CEP is used. The Application Manager Subsystem does not have support on the FIWARE to perform its specific functions. The remaining components have no direct correlation and support from FIWARE and should be implemented from scratch. The VN Manager and Virtualization Sub-system Manager provide management functions that are specific to our proposal. Data Provisioning Manager implements a service specific to our architecture. VN Instances Repository hosts VNs and can be implemented as a data structure in ENs' memory. The Application Entry Point can be implemented as a gateway by handling communication between the CT and ET.

VI. Final Remarks

We proposed a novel three-tier CoS architecture that materializes Olympus virtualization model in a set of software components and extends it by considering the edge tier. Therefore, we leverage the edge tier to provision our VNs closest to the data sources, thus reducing the delay between PSANs and edge devices and contributing to decrease the user´s perceived latency. We discussed how FIWARE can support our system, by putting some implementation effort mainly to decentralize components. We are performing the required changes in FIWARE to generate a full-fledged implementation of our proposed architecture. As future work, we will run experiments with our full-fledged implementation, to assess the performance and energy savings of our approach.

References

[1] A System Architecture for Cloud of Sensors: https://sites.google.com/view/cos-olympus/

[2] A. Chirichiello, "Two Formal Approaches for Web Services: Process Algebras & Action Languages," Sapienza Universita di Roma, 2008.

[3] C. Perera, A. Zaslavsky, C. Liu, M. Compton, P. Christen, D. Georgakopoulos, "Sensor Search Techniques for Sensing as a Service Architecture for the Internet of Things," IEEE Sens. J., vol. 14, no. 2, pp. 406–420, Feb. 2014.

[4] D. Gmach, J. Rolia, L. Cherkasova, A. Kemper, "Resource pool management: Reactive versus proactive or let's be friends," Comput. Networks, vol. 53, no. 17, pp. 2905–2922, Dec. 2009.

[5] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, "Fog Computing and its Role in the Internet of Things," Proc. First Ed. MCC Work. Mob. Cloud Comput., pp.13–16, 2012.

[6] FIWARE: https://www.fiware.org

[7] I. Santos, F. Delicato, L. Pirmez, P. Pires, A. Zomaya, "Resource allocation and task scheduling in the cloud of sensors," in: The Philosophy of Mission-Oriented Wireless Sensor Networks, Eds. Habib Ammari, Springer, 2017 (Accepted for Publication).

[8] I. Santos, L. Pirmez, F. Delicato, S. Khan, A. Zomaya, "Olympus: The Cloud of Sensors," IEEE Cloud Comput., vol.2, no.2, pp.48–56, Mar. 2015.

[9] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, P. Polakos, "Wireless sensor network virtualization: A survey," IEEE Commun. Surv. Tutorials, vol.18, no.1, pp.553–576, 2016.

[10] Implementation of CoS in FIWARE: https://sites.google.com/view/cos-olympus/home/figure1

[11] K. Nishant, P. Sharma, V. Krishna, C. Gupta, K. Singh, Nitin, R. Rastogi, "Load Balancing of Nodes in Cloud Using Ant Colony Optimization," in 2012 UKSim 14th International Conference on Computer Modelling and Simulation, 2012, pp.3–8.

[12] M. Bouzeghoub, "A framework for analysis of data freshness," Proc. 2004 Int. Work. Inf. Qual. informational Syst. - IQIS '04, p. 59, 2004.

[13] M. Yuriyama, T. Kushida, "Sensor-cloud infrastructure-physical sensor management with virtualized sensors on cloud computing," in: 13th International Conference on Network-Based Information Systems," Network-Based Inf. Syst., pp. 1–8, 2010.

[14] Mapping CoS components to FIWARE GEs: https://sites.google.com/view/cos-olympus/home/table1

[15] P. Patel, D. Cassou, "Enabling high-level application development for the Internet of Things," J. Syst. Softw., vol.103, no. C, pp.62–84, May 2015.

[16] R. Eltarras, M. Eltoweissy, "Adaptive multi-criteria routing for shared sensor-actuator networks," GLOBECOM - IEEE Glob. Telecommun. Conf., 2010.

[17] T. Luan, L. Gao, Z. Li, Y. Xiang, G. Wei, L. Sun, "Fog Computing: Focusing on Mobile Users at the Edge," J. Netw. Comput. Appl., vol.52, pp.11–25, Feb. 2015.