A hypervisor for infrastructure-enabled sensing Clouds

Salvatore Distefano*, Giovanni Merlino†‡, Antonio Puliafito†, Alessio Vecchio§

* Dip. di Elettronica, Informazione e Bioingegneria
Politecnico di Milano
Piazza L. Da Vinci 32, 20133 Milano, Italy
Email: salvatore.distefano@polimi.it

† Dip. di Ingegneria
Università di Messina
Contrada di Dio, 98166 Messina, Italy
Email: {gmerlino,apuliafito}@unime.it

‡ Dip. di Ingegneria Elettrica, Elettronica e Informatica
Università di Catania
Viale Andrea Doria 6, 98166 Catania, Italy
Email: giovanni.merlino@dieei.unict.it

§ Dip. di Ingegneria dell'Informazione
Pervasive Computing & Networking Lab (PerLab)
Università di Pisa
Via Diotisalvi 2, 56122 Pisa, Italy
Email: a.vecchio@ing.unipi.it

***Abstract*—The lack of support and the shortcomings of Cloud computing in relation to pervasive applications can be addressed through the Sensing and Actuation as a Service (SAaaS) paradigm. In SAaaS, sensors and actuators, from both mobile devices and sensor networks, can be discovered, aggregated and elastically provided as a service according to the Cloud provisioning model. Nevertheless, managing a large set of sensing and actuation resources, characterized by volatility and heterogeneity, rises the need for specific mechanisms and strategies.**

In this paper we focus on management, abstraction and virtualization of sensing resources. More specifically, we describe the lowest level module of the SAaaS architecture, the hypervisor, that takes care of communication with devices and orchestrates their resources. The hypervisor operates according to policies and strategies coming from higher layers, and includes customization facilities that ease the integration of heterogeneous devices.

I. Introduction and Motivations

Current ICT trends (Future Internet, Internet of Things, etc.) envision a huge number of geographically scattered devices, interconnected through complex and ubiquitous networks. Along the line, mobile Cloud enforces the involvement of sensing and actuation resources in the Cloud, mainly as endpoints. In [1] we reversed such perspective, considering sensors and actuators in the same way as computing and storage resources are considered in more traditional Cloud stacks: abstracted, virtualized and grouped in Clouds. We proposed the "Sensing and Actuation as a Service" (SAaaS), a provisioning model where (virtual) sensors and actuators are offered as a service, in an IaaS fashion, to users.

In SAaaS, contributing nodes can be either static or mobile, "voluntarily" shared by their owners or administrators, thus they can join and leave the system according to unpredictable patterns, according to their owners or administrators will.

Partially funded by PhD programme under grant PON R&C 2007/2013 "Smart Cities" and by Simone project under grant POR FESR Sicilia 2007/2013 n. 179.

Previous research mostly followed a data-driven approach, where the focus was on the integration of information produced by sensors with Cloud-based applications, posing some unneeded restrictions as consumer applications cannot customize collection/actuation operations according to their needs. In our vision sensing Clouds should provide virtual devices that can be manipulated by requesting applications, not just a representation of the data they produce/consume.

With our device-driven approach, sensing applications have total control of the allotted virtual sensors and actuators through their handlers or APIs. For example, a wired/wireless sensor network (SN) owner whose sensing infrastructure does not adequately cover a specific area of interest may inquire an SAaaS provider for additional sensors/actuators. The latter will provide the requested resources, i.e. virtual sensors/actuators, that can be incorporated into the requester's network in a cloudbursting fashion. The requester can also partially customize the behavior of obtained resources, if needed.

This paper describes an architecture that provides the fundamental mechanisms for abstraction, virtualization, and management of resources. The proposed solution is designed to fit within the SAaaS architecture defined in [1], in particular the SAaaS *Hypervisor* module. The term Hypervisor has been used to highlight the analogies between SAaaS and IaaS: a VMM manages computing resources under the guise of virtual machines; in the same way the SAaaS Hypervisor manages virtual sensors and actuators.

In the sensor technology domain virtualization has been proposed to enable seamless interoperability and scalability of sensor node platforms from different vendors, with the insertion of an abstraction layer between the application logic and the sensor-driver [2]–[4]. Software abstraction layers are used to address interoperability and manageability issues [5]–[8], to enable the dynamic reconfiguration of sensor nodes [9], [10], and fusion of sensor data [11], [12].

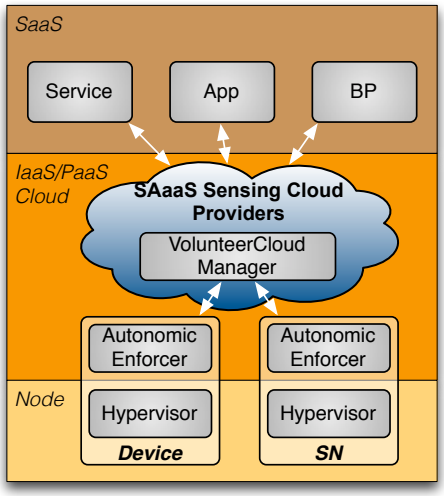

Figure 1. Architectural schema and modules

This paper proposes an architecture that enables devices to provide their sensors as virtualized instances, possibly featuring a subset of functionalities of the underlying physical devices, thus multiplexing concurrent requests for sensing devices on the same physical one. Another goal consists in generating brand new resource instances, in the sense of repurposing (networks of) devices, by means of composition and aggregation of simpler ones. Additionally, the proposed architecture includes mechanisms for customizing and contextualizing the sensing resources. To the best of our knowledge there's no existing literature addressing the aforementioned issues from a device-oriented perspective. Indeed, most of previous research followed the data-driven approach, i.e. exposing and managing data rather than devices.

II. Overview of SAaaS sensing Cloud

SAaaS is a paradigm aimed at developing a sensing infrastructure based on sensors and actuators from both mobile devices and SNs, in order to provide virtual sensing and actuation resources in a Cloud-like fashion. More specifically, it aims at delivering the basic mechanisms and tools for enabling a Cloud of sensors and actuators, which have to be adequately extended and customized by providers in order to implement enhanced services and provisioning models. To this end, the main issues that have to be faced include: abstraction of sensing and actuation resources, virtualization, customization, monitoring, SLA and QoS management, subscription and policy management, enrollment, indexing and discovery, security and fault tolerance.

This fosters the design of a software stack that implements the following main functionalities: i) involvement of SNs, smartphones or other devices endowed with sensors and/or actuators, and their enablement for interoperation in a Cloud environment; ii) distributed mechanisms and tools for self-management, configuration and adaptation of nodes; iii) functions and interfaces for enabling and managing resources that are voluntarily contributed.

In order to build such ambitious idea, i.e. a Cloud of Sensors based on the SAaaS paradigm, in [1] we introduced the whole stack and a rough schema of the architectural modules. The three main components of the architecture are shown in Fig. 1, bottom-up: Hypervisor, Autonomic Enforcer and VolunteerCloud Manager.

The term SAaaS *node* is used to indicate a smart device equipped with sensors, such as a smartphone, or a frontend to a possibly large number of small sensing devices, such as the gateway of a SN. In other words, in case of relatively powerful devices, the device itself is the SAaaS node, while, in case of SNs, the SN gateway is identified as the SAaaS node. SAaaS *mote* is instead the elementary hardware block of the sensing Cloud infrastructure. In SNs, a mote is a programmable sensing board equipped with computing and networking resources, able to perform processing, routing or storage operations. With regards to devices such as smart-phones or PDA, the term is used to indicate the characterization, and possibly specialization, of the sensors and actuators available on-board. Through the SAaaS mote concept it is therefore possible to logically group the sensing resources provided by a device, thus unlocking a further degree of freedom in SAaaS resource management. It is important to remark that in Wireless SN terminology the terms "node" and "mote" are usually used as synonyms, but in SAaaS we use them with different meanings.

The lowest block, the Hypervisor, operates at the level of a single node, where it abstracts the available sensors. The main duties of the Hypervisor are: communications and networking, abstraction of devices and capabilities, virtualization of abstracted resources, customization, isolation, semantic labeling and thing-enabled services. These features and the most relevant issues are discussed in the following section.

The Cloud modules, under the guise of an Autonomic Enforcer and a VolunteerCloud Manager, deal with issues related to the interaction among nodes. The former is responsible of the enforcement of Cloud policies, local vs. global policy tie-breaking, subscription management, cooperation on overlay instantiation. Whenever possible it operates according to autonomic principles. The latter is instead in charge of the following functionalities: exposing the Cloud of sensors via Web Service interfaces, indexing of resources, monitoring of Service Level Agreements (SLAs) and Quality of Service (QoS) metrics.

These layers thus form a coupled, two-level Cloud stack, where many mechanisms are split in both modules, dealing with node-wise actions and self-organizing properties in the lower one, and centrally managed Cloud-wise methods in the upper one.

III. The Hypervisor

The Hypervisor is the fulcrum of a device-driven approach to infrastructural Clouds of sensors: it manages the resources dedicated to sensing and actuation, introducing abstraction and virtualization functionalities. It operates at the level of the single SAaaS node, i.e. either an entire SN under the administration of a single authority, or a standalone/set of sensors within a device such as a smartphone. As discussed in Section II, in SNs an SAaaS node may be less easily

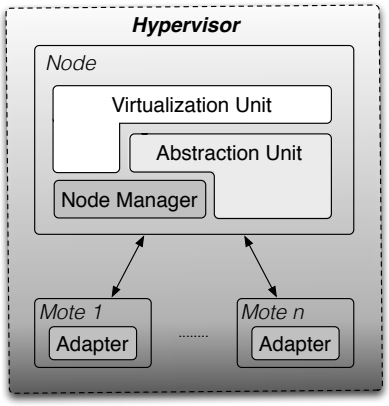

Figure 2. The Hypervisor module architecture.

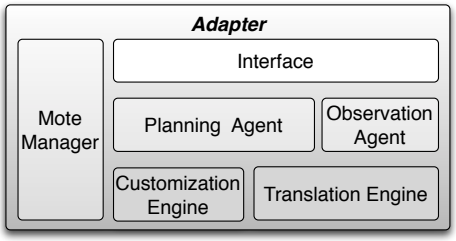

Figure 3. Adapter modules.

identifiable with respect to the one-to-one relationship we have for smartphones and other personal devices: we may have a SN made up of thousands of motes, yet only exposing few base stations, where nodal components of the Hypervisor can be deployed, in contrast to the lowest levels of the Hypervisor for the aforementioned motes.

In order to deal with the above mentioned issues, a modular architecture of the Hypervisor has been specified. The architecture, depicted in Fig. 2, identifies four major components: Adapter, Node Manager, Abstraction Unit and Virtualization Unit, spanning among nodes and corresponding motes. The reasons for splitting the Hypervisor between the node (e.g. gateway/base station) and the motes mainly lies in the requirement, embodied by our device-driven approach, to offer near-complete access to sensing resources. Such splitting is also a consequence of the autonomic management of distributed resources (as it is desirable with complex and large SNs).

A. Adapter

At the lowest level of the stack for the Hypervisor lies the Adapter, a modular diagram of which is depicted in Fig. 3. As the component of the Hypervisor running at the edge (e.g. on a mote), the Adapter has a three-fold role.

Firstly, it exposes a standard-compliant (e.g. SensorML) customer-friendly (e.g. RESTful) *Interface* to on-board resources. The *Observation Agent* requests, retrieves and eventually pre-processes measurements. The *Planning Agent* sends commands to the device for the management of operating parameters such as duty cycle, sampling frequency, etc. Both rely on the presence of a platform-specific driver, located within the *Translation Engine*, for converting high-level directives into native commands. Moreover, this approach for interacting with the underlying resources features isolation as a built-in, a requirement for proper infrastructure virtualization.

Secondly, the Hypervisor deals with the demand for deep (provider-induced) reconfiguration of device functionalities by means of the *Customization Engine*, an interpreter able to execute on the sensing device the code needed to tailor the sensing activities to the specific needs. Let us suppose, for instance, that the readings provided by a sensor are characterized by some noise and that some form of filtering, such as median filtering, is needed. The Hypervisor can receive from the Cloud the specifications of the median-filtered sensor and the code that transforms the raw readings according to such specifications. By sending these code fragments, the client applications running in the Cloud can customize the sensing (and actuation) process directly on the device. This makes possible the customization of the sensing activities according to unforeseen criteria, e.g. to compress the data before sending them to the Cloud and thus saving bandwidth and energy. The same approach is followed to add new sensing functionalities on a device. To provide an additional example, let us suppose that a client application needs a sensor that detects human presence and that presence can be inferred by properly combining the level of sound registered by the microphone and movement acquired through accelerometers. In such case, a new virtual presence sensor can be realized by uploading onto the device a document containing both the description of the new sensor and the set of operations that have to be performed on sound and acceleration values to obtain the desired results. Then the new presence sensor is also going to be registered into the Cloud indexing service. At last, the Adapter takes care of managing the whole set of devices inside a domain according to an autonomic approach. This enables power-driven self-optimization and other self-configuration mechanisms (e.g. networking). To this purpose, the elements of the Adapter located on the base station cooperate with the *Mote Manager*, which is executed on the mote-side. Given the widespread and cheap availability of smart boards for motes, pushing intelligence to the edge of the SN is easy to achieve. In any case, dumb sensing platforms could still be considered for inclusion through adequate proxy-based mechanisms. The Planning Engine is responsible for the interaction between customers and the Mote Manager, driving contextualization requests over virtualized resources only through the restricted subset of available customization actions exposed by the Customization Engine. Dealing with networking issues is out of the scope of this work, yet world-wide addressing and application-level communications are either partially solved issues (e.g. Cloud2Device messaging) or pertinence of standardization efforts (e.g. ETSI M2M). In terms of security, among the issues to be tackled in future efforts, remote attestation of motes is of particular importance.

Moreover, meta-data processing within the volunteer Cloud must respect privacy of any personal trails that are related to virtualized nodes. This requirement must hold even if this kind of data is not managed by the volunteer Cloud, but is just passed along, from the device to the customer. All of the above holds true also with smartphones, with the obvious difference

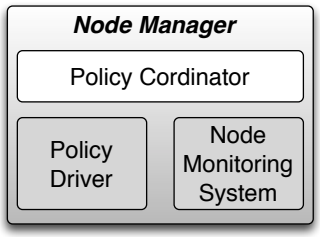

Figure 4. Node manager modules.

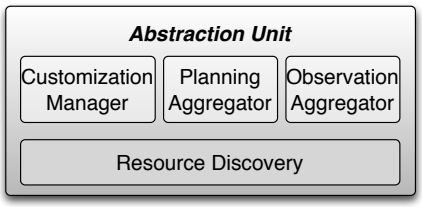

Figure 5. Abstraction unit modules.

that being there one board/platform only per mobile, there would be just one Adapter correspondingly.

Our research team has started addressing the lowest component of the Hypervisor, the Adapter, with additions to an in-house developed IaaS stack, characterized by a flexible approach to inter-node/Cloud communication and event notification [13], a fork planned to work over a common baseline, having chosen the TinyOS and Contiki open source platforms.

B. Node Manager

The *Node Manager* is in charge of sensing resource operations, as well as enforcement of policies, just at node level.

In particular, the *Policy Coordinator* takes care of resolving conflicts between the set of Cloud-wise policies and local ones (e.g. imposed by the owner or the administrator). The *Policy Driver* implements the actual enforcement of sensing and actuation policies as relayed by the Coordinator. Policies are translated into rules to be applied on combos coming from other modules (in detail, the Planning Aggregator inside the Abstraction Unit), as well as into constraints to be enforced on power self-tuning activities. Moreover, here is where user-mandated policies on resource management get stored.

These node-level policies and strategies are then enforced at the mote-level through the Policy Driver. In particular, on the mote side, policies are receipt by the Adapter's Mote Manager which, as mentioned, operates according to autonomic principles. In other terms, the coordinator elaborates policies and strategies that are either enforced by the driver if they do not involve motes, or deployed by the driver into the motes and enforced by the corresponding Mote Managers in a self-managing, autonomic fashion.

The Node Manager can also (optionally) include a *Monitoring System* that, in collaboration with the Adapter Observation Agent, provides historical data and statistics. This provides the opportunity of evaluating the effectiveness of runtime strategies. This information can be also provided to higher level modules, such as the Observation Aggregator of the Abstraction Unit described below.

C. Abstraction Unit

At a very basic level, the Abstraction Unit replicates planning and observation facilities modeled after those featured in the Adapter but on a node-wide scale, combining the pool of resources of the whole SN or smartphone, as shown in Fig. 5. The *Observation Aggregator* exposes the set of node-wide resources, and the *Planning Aggregator* manages this set, sending combos, i.e. combination of commands, and tracking exit codes, eventually reacting to (partial) failures by triggering apt adjustments.

At the bottom lies the *Resource Discovery* module, which offers an interface to the motes, actively gathering descriptions of underlying resources and forwarding the results to the Aggregator modules.

Finally, the *Customization Manager* acts as an orchestrator for the customization engines located on the motes. In fact, depending on the type of customization, the code is executed on the SN gateway or on the remote nodes. In more detail, customization is carried out on motes when such activity is compatible with their HW limitations, e.g. when computing median-filtering, calibration, or simple forms of compression, whereas customization is executed on the gateway when it requires the output of multiple sensors or when a substantial amount of computing power is required.

If the observations produced by sensors are encoded using an XML-based dialect, such as *Observations&Measurements*, then customization could be based on XML extensible stylesheet language transformations (XSLT). XSLT is a language for transforming XML documents into other XML ones. In the XSLT language, a transformation is expressed in the form of a stylesheet, a well-formed XML document, and it is executed by an XSLT processor. It has also been demonstrated that XSLT is a Turing-complete language. In the context of the SAaaS architecture, XSLT can be used by client applications to specify the customization actions that the application needs to execute on the enrolled sensing devices. If an application wants to perform some noise reduction on a given sensor, the former may prepare an XSLT document that implements, for instance, filtering by computing the average. Then the XSLT document is transferred onto the node, where it is consumed by the Customization Manager.

Since transformations are expressed as XML documents, the developer of the client application is not forced to use some programming language. Moreover, code transformations are portable across different architecture and implementation of sensing devices, as long as a XSLT processor is available for such configuration. Implementation of a XSLT processor is possible on resource-rich devices, e.g. smartphones. For resource-constrained devices, XSLT transformations can still be operated on the node (SN gateway). However, the Customization Manager may decide to push part of the customized processing out to motes. In this case, the Customization Manager should isolate specific portions of code and send them to its slaves. Given the constraints imposed by the limited resources available, mote-side customization may not be based

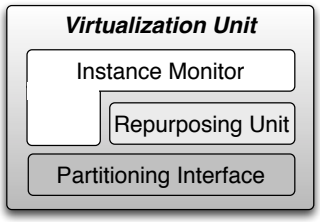

Figure 6. Virtualization Unit modules.

on portable code and should then be adapted to the specific hardware and software platform. For example if the mote is based on TinyOS, customization can be implemented through Adaptlets [10], small code fragments to be executed on motes by an application adaptation layer.

D. Virtualization Unit

In Fig. 6 we have the modules belonging to the Virtualization Unit, whose core activity is *slicing*, i.e. generating possible partitioning schemes for the cluster of resources exposed by the Abstraction Unit. These partitioning schemes can be subsequently ranked according to a number of criteria including sensor provenance, proximity, QoS, security and so on. In detail, slicing and ranking are carried out by the *Partitioning Interface*. We believe that this is quite a novel theme in this context, not featured in classic IaaS.

Note that matching between customer requests and availability of resources is operated by the VolunteerCloud Manager. The latter takes arrival order, pre-existing choices and other constraints into account, together with the slicing regenerated at every occurrence of triggering events such as addition or removal of sensing infrastructure.

The *Repurposing Unit* provides a wider variety of virtual sensors with respect to the underlying resources. This involves choosing certain slicings with specific sets of devices, or computing transformations on measurements, to shape the functionality the node exports. The result is a set of sensors/actuators in the widest possible meaning. For example a weather sensor, can be offered by combining other simpler sensors such as temperature, pressure and humidity. Repurposing can be pushed further, e.g. a camera with motion detection software to replace missing positioning sensors.

At last, we envisioned the *Instance Monitor* as the upper building block of the Virtualization Unit, named after the Virtual Machine Monitor to highlight its role as a manager of the lifecycle of virtualized resource instances. This includes APIs and functionalities for virtual instance creation and reaping, as well as for boot- (defined statically) and run-time (dynamically) parameters discovery and tuning in accordance to contextualization requests.

IV. Conclusions

The idea of combining sensing resources and Cloud infrastructure is as appealing as it is challenging. SAaaS aims at bridging the gap between sensing (and actuation) pervasive scenarios and service-oriented provisioning models. At the lowest level of the SAaaS stack, the Hypervisor allows to manage, abstract and virtualize sensing and actuation resources that could be provided by enrolling either mobile devices or SNs, in a volunteer contribution fashion. This paper provides an architectural solution covering several unique features of the SAaaS Hypervisor, including customization facilities for adapting physical resources to their virtual instances .

References

[1] S. Distefano, G. Merlino, and A. Puliafito, "Sensing and actuation as a service: A new development for clouds," in *Proceedings of the 2012 IEEE 11th International Symposium on Network Computing and Applications*, ser. NCA '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 272–275. [Online]. Available: http://dx.doi.org/10.1109/NCA.2012.38

[2] M. Iqbal, D. Yang, T. Obaid, T. J. Ng, and H. B. Lim, "Demo abstract: A service-oriented application programming interface for sensor network virtualization," in *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, april 2011, pp. 143 –144.

[3] S. Alam, M. Chowdhury, and J. Noll, "Senaas: An event-driven sensor virtualization approach for internet of things cloud," in *Networked Embedded Systems for Enterprise Applications (NESEA), 2010 IEEE International Conference on*, nov. 2010, pp. 1 –6.

[4] H. B. Lim, M. Iqbal, and T. J. Ng, "A virtualization framework for heterogeneous sensor network platforms," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '09. New York, NY, USA: ACM, 2009, pp. 319–320. [Online]. Available: http://doi.acm.org/10.1145/1644038.1644080

[5] M. Navarro, M. Antonucci, L. Sarakis, and T. Zahariadis, "Vitro architecture: Bringing virtualization to wsn world," in *Proceedings of the 2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*, ser. MASS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 831–836. [Online]. Available: http://dx.doi.org/10.1109/MASS.2011.96

[6] M. Yuriyama and T. Kushida, "Sensor-cloud infrastructure - physical sensor management with virtualized sensors on cloud computing," in *Network-Based Information Systems (NBiS), 2010 13th International Conference on*, sept. 2010, pp. 1 –8.

[7] G. Hackmann, C.-L. Fok, G.-C. Roman, and C. Lu, "Agimone: middleware support for seamless integration of sensor and ip networks," in *Proceedings of the Second IEEE international conference on Distributed Computing in Sensor Systems*, ser. DCOSS'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 101–118. [Online]. Available: http://dx.doi.org/10.1007/11776178_7

[8] M. J. Ocean, A. Bestavros, and A. J. Kfoury, "snbench: programming and virtualization framework for distributed multitasking sensor networks," in *Proceedings of the 2nd international conference on Virtual execution environments*, ser. VEE '06. New York, NY, USA: ACM, 2006, pp. 89–99. [Online]. Available: http://doi.acm.org/10.1145/1134760.1134774

[9] A. P. Jayasumana, Q. Han, and T. H. Illangasekare, "Virtual sensor networks - a resource efficient approach for concurrent applications," in *Information Technology, 2007. ITNG '07. Fourth International Conference on*, april 2007, pp. 111 –115.

[10] M. Avvenuti, P. Corsini, P. Masci, and A. Vecchio, "An application adaptation layer for wireless sensor networks," *Pervasive Mob. Comput.*, vol. 3, no. 4, pp. 413–438, Aug. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.pmcj.2007.04.001

[11] K. Aberer, M. Hauswirth, and A. Salehi, "Infrastructure for data processing in large-scale interconnected sensor networks," in *Mobile Data Management, 2007 International Conference on*, may 2007, pp. 198 – 205.

[12] Y. Liu, D. Hill, A. Rodriguez, L. Marini, R. Kooper, J. Myers, X. Wu, and B. Minsker, "A new framework for on-demand virtualization, repurposing and fusion of heterogeneous sensors," in *Proceedings of the 2009 International Symposium on Collaborative Technologies and Systems*, ser. CTS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 54–63. [Online]. Available: http://dx.doi.org/10.1109/CTS.2009.5067462

[13] M. Fazio, M. Villari, and A. Puliafito, "Sensing technologies for homeland security in cloud environments," in *Sensing Technology (ICST), 2011 Fifth International Conference on*, 28 2011-dec. 1 2011, pp. 165 –170.

[14] B. Caprarescu, N. M. Calcavecchia, E. D. Nitto, and D. J. Dubois, "SOS Cloud: Self-Organizing Services in the Cloud," in *Bionetics '10: Bio-Inspired Models of Network, Information and Computing Systems*, 2010.