2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS): SmartCity16: The 2nd IEEE INFOCOM Workshop on Smart Cities and Urban Computing

Stack4Things as a fog computing platform for Smart City applications

Dario Bruneo*, Salvatore Distefano*‡, Francesco Longo*, Giovanni Merlino*, Antonio Puliafito*,
Valeria D'Amico†, Marco Sapienza†, Giovanni Torrisi†
**Università di Messina, Italy*
{*dbruneo,sdistefano,flongo,gmerlino,apuliafito*}*@unime.it*
†*Telecom Italia, Italy*
{*valeria1.damico,marco1.sapienza,giovanni.torrisi*}*@telecomitalia.it*
‡ *Kazan Federal University, Russia*
sdistefano@kpfu.ru

***Abstract*—Fog computing envisions computation logic to be moved at the edge of the Internet where data needs to be quickly elaborated, decisions made, and actions performed. Delegating to the Cloud the whole burden of applications could not be efficient indeed, for example in case of workload bursts. This is especially true in the context of IoT and Smart City where thousands of smart objects, vehicles, mobiles, people interact to provide innovative services. We thus designed Stack4Things as an OpenStack-based framework spanning the Infrastructure-as-a-Service and Platform-as-a-Service layers. It enables developers and users to manage an IoT infrastructure, remotely controlling nodes as well as virtualizing their functions and creating network overlays among them, implementing a provisioning model for Cyber-Physical Systems. Moreover, it provides mechanisms to scatter the application logic on top of the involved smart objects and to choose with fine granularity which specific tasks to delegate to centralized Cloud infrastructure. In this paper, we show the core Stack4Things mechanisms implementing a Fog computing approach towards a run-time "rewireable" Smart City paradigm. We demonstrate its effectiveness in a smart mobility scenario where vehicles interact with City-level smart objects to provide end users with highly responsive geolocalised services.**

***Keywords*-Stack4Things; OpenStack; fog computing; function virtualization; code injection; plugin-based development; AllJoyn.**

I. Introduction

The Smart City scenario is a fertile application domain for different sciences, in particular for the information and communication ones indeed, as also confirmed by ongoing projects [1] highlighting the need to leverage Cyber-Physical Systems (CPS) as Smart City facilities. From this perspective, cities may be regarded as complex "ecosystems" composed of heterogeneous interconnected CPS providing sensing and actuating facilities, such as traffic sensors, security cameras, traffic lights as well as citizen smartphones. On the other hand, from a high level perspective, new services and applications can be envisioned, from traffic monitoring to energy management, from e-health to emergency management. This scenario calls for novel solutions to manage the underlying physical sensing and actuation resource infrastructure and enable new services, bridging the gap between the application level and the Smart City CPS ecosystem one.

The research discussed in this paper has been conducted within a research agreement with TIM Joint Open Lab WAVE on the "Study, definition and development of a Sensor Cloud system based on the SAaaS paradigm".

A possible solution could be to decentralize the processing to the edge where data is generated, an approach recently defined as Edge or Fog computing. The term *Fog computing* has been introduced by Cisco [2] as an extension of Cloud computing to the edge of the Internet dealing with mobility, heterogeneity, very large number of nodes while improving performance and extending battery life of resource-constrained/mobile devices by offloading some computation to a second level Cloud infrastructure characterized by proximity and low-latency (e.g., *Cloudlet* [3]). The main use cases in this direction are distributed analytics [4], data mining [5], and situation awareness [6].

In this work, we push Fog computing to an even lower logical level as a set of mechanisms enabling the deployment and execution of multiple location-aware, low-latency, peer-to-peer IoT-like applications, where the greatest part of the logic runs on the involved smart objects. This way, the centralized Fog infrastructure acts as a programmable coordinator only, able to detect specific developer-defined situations and to actuate multiple actions, including modifications in the infrastructure topology where needed. As soon as such actions are triggered, the Fog coordination subsystem acts only as a tunneling and forwarding agent enabling inter-object communication at data-link/network layers whereas no involvement at the application layer is needed. To this purpose, we base our solution on the Sensing and Actuation as a service (SAaaS) approach [7], aiming at providing sensing and actuation resources as a service, on-demand, adopting a Cloud-oriented model to transducer-related resource provisioning. We are implementing this approach in the Stack4Things middleware [8], adapting and extending OpenStack to CPS infrastructure.

Some Fog computing frameworks and IoT-Cloud computing platforms are available in the literature. For example, the oneM2M architecture and its application to the management of connected vehicles is discussed in [9], albeit this architecture does not envision the exploitation of CPS also as so-called *middle* nodes, e.g., M2M gateways. In [10], authors first provide a set of principles for engineering IoT-Cloud systems and then introduce their iCOMOT prototype. The definition of the FemtoCloud system and some experimental results are provided in [11] highlighting its ability to leverage the available device processing capacity. The main differences between Stack4Things and the above referred frameworks are: (i) the Stack4Things architecture is built on top of OpenStack that is the de-facto standard framework for building Infrastructure-as-a-Service Clouds; (ii) CPS devices are considered both as things and as part of the edge, e.g., providing Cloud services on demand; (iii) we provide an open source implementation[1]; (iv) Stack4Things is adopted in the #SmartME project[2] where hundreds of smart objects are disseminated throughout the city of Messina.

The main contribution of this paper is therefore fourfold: i) a new approach to Fog computing pushing intelligence to devices, based on ii) a novel CPS functions virtualization conceptual framework, iii) rewiring mechanisms for reconfigurable Smart City, and iv) contextualization mechanisms for specializing Smart City devices by injecting custom application-level code. These artifacts are firstly introduced, from a methodological point of view in Section II and then discussed in practical terms as facilities of the Stack4Things middleware in Section III. A use case on a smart mobility application is described in Section IV to demonstrate the effectiveness of the approach providing a quantitative evaluation. Final remarks in Section V close the paper.

II. Approach to Reconfigurable Smart Cities

To manage heterogeneous and complex socio-technical systems on the scale of whole cities, where both social and technological issues merge, an overarching approach is required. Specifically, on the one hand the goal is to provide a uniform representation of connected smart objects (more generally CPS) by abstracting, grouping, and managing them as a unified ecosystem of *objects* to be configured and contextualized according to application requirements. On the other hand, a management layer able to control the dynamics of the ecosystem, able to map such requirements into lower level ones, implementing and enforcing specific policies to satisfies such requirements is needed.

A suitable solution may therefore lie in adopting a "rewiring" approach, where basic mechanisms are provided by a centralized infrastructure driving smart cities objects, to *group* objects together at the lowest level achievable in terms of the networking stack, letting them interact with each other, as well as specialize their behavior.

[1]Stack4Things is available online at *stack4things.unime.it*.
[2]For more information about #SmartME refer to *smartme.unime.it*.

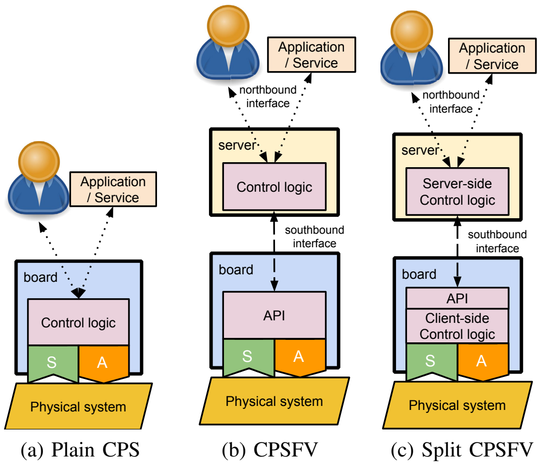


Figure 1. Cyber-Physical System: Functions Virtualization (CPSFV)

In Figure 1 we identify three options to sketch a simple instance of a generic CPS, i.e., one that includes a single smart interface interacting by means of its transducers (sensors and actuators) with a physical entity. The standard configuration, as depicted in Figure 1a, features a "plain" CPS, thus the interface subsystem (e.g., a board) acts on its own, and any end-user or application interacts with the physical world through the node itself. A first useful abstraction is proposed in Figure 1b, where the role played by the interface gets (partially) shifted from the actual hardware instance of the sensor-/actuator-hosting platform to another (physically detached, possibly remote) machine, whose only requirements are some available processing (and storage) quotas. This means exposing a *northbound* interface mostly equivalent to the one provided by a plain configuration, leaving to a *southbound* interface to expose just low-level I/O primitives. To capture this notion, this configuration may be aptly labeled as *CPS Functions Virtualization* (CPSFV). A further development of this idea is shown in Figure 1c, which synthesizes both plain and virtualized approaches into a hybrid, split CPSFV one, where some control logic could be deployed both in a remote server and/or in the board.

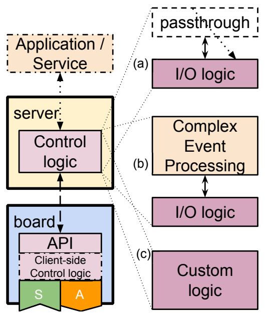


Figure 2. Control logic: approaches.

Zooming in on the virtualization blocks, we may envision a series of approaches to the design of control logic as empowered by function virtualization capabilities. These are shown in Figure 2 in the split CPSFV case as a generalization that collapses into the simple CPSFV case if the client control logic is not present. At a minimum there should be the option to just get passthrough access, as depicted in Figure 2a, from the application directly to I/O logic, to just let the virtual board drive the transducers and leave all other duties to the application itself. Shifting more duties from the application level to the (virtual) infrastructure may be obtained by injecting (a part of, or the whole application-level) control logic as rules for a *Complex Event Processing* (CEP) engine to consume and act upon, as described in Figure 2b. In the end, the developer may skip this abstraction altogether and just inject some custom code, featuring both the rules and any I/O driving logic or abstraction layer, as highlighted in Figure 2c.

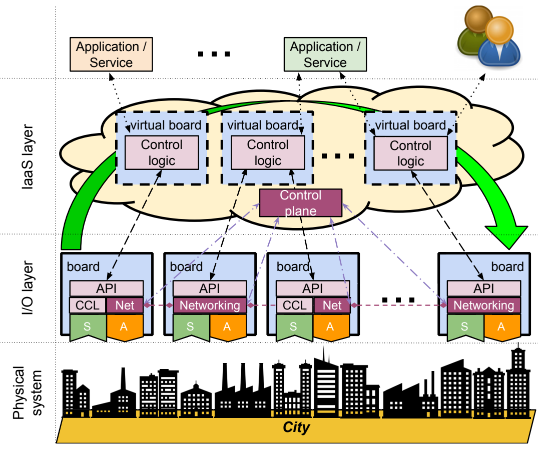


Figure 3. Rewireable Smart City.

In line with our overarching CPS virtualization approach, such foreseeable outcome ultimately calls for function virtualization at a metropolitan scale, a reconfigurable (*rewireable*) Smart City, depicted in Figure 3, as a set of CPS where Cloud-hosted *virtual boards* are introduced to represent virtual instances of servers (under the guise of VMs) acting in place of physical devices. This way, we may identify two "cyber" levels above the large-scale physical system that is the City, identifying a City-wide CPS altogether. A distributed *I/O layer* and a centralized subsystem modeled as an *Infrastructure-as-a-Service layer*, also extending the approach to include dynamic reconfiguration of the networking subsystem of the underlying nodes. To this purpose, a Software Defined approach could be adopted to manage the City-scale (networking) infrastructure, thus introducing in the high-level architecture a Control plane at IaaS layer and basic (networking/forwarding) functionalities at I/O layer (data plane) as shown in Figure 3. Thereby, a generic capability to inject code on physical boards is available, unlocking a degree of freedom in the customization of smart objects and thus enabling edge-oriented approaches towards fog computing, also enabling infrastructure-level customization, such as board-side virtual networking facilities.

Furthermore, a middleware devoted to the management of both sensor- and actuator-hosting resources may help in the establishment of higher-level services, including policies for "closing the loop", e.g., configuring triggers for a range of (dispersed) actuators based on sensing activities from (geographically non-overlapping) sensing resources. A high-level depiction of that is also reported in Figure 3 (green arrow), where these mechanisms are there to *rewire* such a "nervous system" into a number of elastic control loops.

III. Implementation and core mechanisms

In the pursuit for integration of CPS infrastructure with paradigms and frameworks for geographically dispersed resource management, Cloud computing facilities, here also implementing a service-oriented approach in the provisioning and management of sensing and actuation resources, are leveraged to enable a device-centric [12] SAaaS paradigm for Fog-like exploitation of CPS resources. In fact, in the SAaaS perspective, sensing and actuation-enabled systems should be handled along the same lines as computing and storage abstractions in traditional Clouds, i.e., on the one hand virtualized and multiplexed over (scarce) hardware resources, and on the other grouped and orchestrated under control of an entity implementing high level policies. This enables our Fog approach by empowering the user to push custom code even to devices at the edge of the complex CPS that is the Smart City, as well as to throw those devices in a shared pool to enable machine-to-machine interactions. This way, I/O endpoints and their backing subsystems have to be part of the Cloud infrastructure and have to be managed by following the consolidated Cloud approach, i.e., through a set of APIs ensuring remote control of software and hardware resources despite their geographical position.

A Cloud-oriented solution may fit CPS-powered Fog scenarios indeed, meeting most requirements by default to cater to the originally intended user base while, at the same time, also addressing other more subtle functionalities, such as a tenant-based authorization framework, where several actors (owners, administrator, users) and their interactions with infrastructure may be fully decoupled from the workflows involved (e.g., transfer, rental, delegation). This way, our CPSFV-enabled prototype, Stack4Things, is based on the OpenStack IaaS middleware. Being geared towards embedded systems, Stack4Things adopts a leaner approach, i.e., in the design and implementation of the board-side components, both in terms of messaging [13], as well as trading off full-blown VMs for custom logic to be instantiated under the guise of plugins, or within containers at most, leaving more heavyweight mechanisms to the Cloud-side subsystems only.

A. Cloud-based virtualized networking

The basic remoting mechanisms are based on the creation of generic TCP tunnels over WebSocket (WS), a way to get client-initiated connectivity to any server-side local (or remote) service. In this sense, we devised the design and implementation of an incremental enhancement to standard WS-based facilities, i.e., a *reverse* tunneling technique, as a way to provide server-initiated, e.g., Cloud-triggered, connectivity to any board-hosted service.

Beyond mere remoting, level-agnostic network virtualization needs mechanisms to overlay network- and datalink-level addressing and traffic forwarding on top of such a facility. In [13] the authors describe a model of tunnel-based layering employed for Cloud-enabled setup of virtualized bridged networks among boards across the Internet.

B. Contextualization

Here we define *contextualization* as Cloud-enabled injection of custom code on any board at runtime under the guise of independent pluggable modules, i.e., a way to adapt the behavior of the CPS edge device under consideration to the task at hand (context). The runtime environment of choice in our case is Node.js, a modern Javascript runtime meant for server-side development, thus featuring advanced programming facilities, very high extensibility, a huge collection of third-party libraries and a comprehensive OpenSource ecosystem overall.

The main components implemented in the board-side Stack4Things engine for contextualization are a plugin management library, as well as a default wrapper meant for plugins to be isolated (as processes to be spawned by the engine) and to bring their lifecycle under control. This way, any Stack4Things user is able to upload and inject her own plugin, still according to authorization and privacy policies, while also mediating access conflicts. The injection process is carried out by transferring the code as payload of a WAMP [14] RPC message, and registering the plugin among the available ones for the involved boards. An instantiated plugin can be defined in this context as any process running in parallel that executes a workflow, e.g., a continuous collection of sensor measurements. It is essential to remark that this definition does not exclude more critical operations, which may be part of the customization at runtime as well, and thus delegated to plugins as well, such as the board-side virtual networking subsystem.

In particular the following methods have been implemented to manage plugins on a board:

- *runPlugin* - to manage the execution of a plugin.
- *killPlugin* - to execute the procedure of termination of a plugin.
- *injectPlugin* - to manage the process of injection of a plugin on a board.
- *restartAllActivePlugins* - to automatically restart at boot time all the plugins that have been selected by the user or that have to be restored after a failure of the board-side Stack4Things engine.

The steps of the process for plugin activation are the following: (i) the user registers the source code of the new plugin in the Stack4Things Cloud, specifying the schema of the input parameters and configurations using the Web-based dashboard; (ii) the user deploys (injects) the new plugin onto the board of interest specifying input parameters; (iii) the user activates the plugin already deployed onto the board.

On the Cloud-side, the management of the plugins is split into multiple actions, as in fact it is possible to create, inject, run and kill a plugin, respectively, that is:

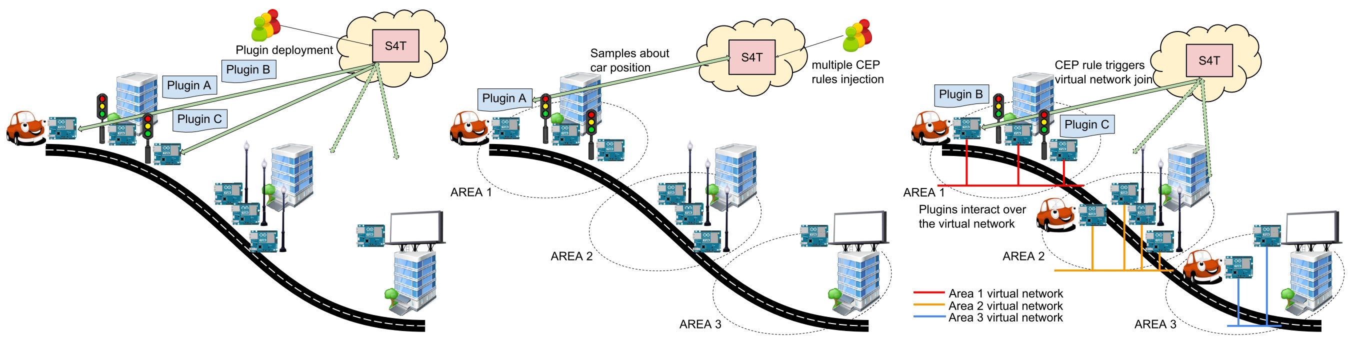
- *create* - storing a plugin on the Cloud by specifying the name, the JSON-encoded schema parameters and the raw code (it could be also a file). Once sent to the Cloud this plugin is available for any board that wants to use it;
- *inject* - installing the plugin onto the board-side Stack4Things subsystem of the selected board. Furthermore it is possible to choose if the plugin should run at boot time;
- *run* - launching the plugin;
- *kill* - killing the process corresponding to a plugin instance, whenever it is necessary to stop a running task.

IV. A SMART MOBILITY USE CASE

To demonstrate the concepts and functionalities described so far in a Smart City setting, we designed a smart mobility use case. It aims at showing the effectiveness of the Stack4Things platform in: i) allowing the developer to dynamically build/destroy Cloud-mediated virtual networks; ii) providing mechanisms for the deployment of a set of Node.js-based plugins on the smart objects; iii) supplying the developer with the possibility to deploy a part of the application logic also at the Cloud level by injecting a set of CEP rules; iv) allowing the developer to leverage legacy and/or proprietary platforms with specific and strict low-level requirements while designing and implementing her application.

A. Use case description

A Smart City developer designs a distributed application providing the user with a set of geolocalized services deriving from the interaction of her smart car with several smart city objects, including, e.g., smart traffic lights, smart streetlights, smart advertising billboards. Examples of such services can be: (i) allowing a smart car to interact with a set of smart traffic lights to acquire a certain level of priority with respect to other cars while approaching an intersection; (ii) at night time, enabling a vehicle to signal its presence so that smart streetlights installed along the road may ramp their brightness up for driving safety reasons; (iii) allowing a



(a) Code injection (b) CEP triggering (c) VPN insert and discovery

Figure 4. The three phases of the use case.

smart billboard to be populated with personalized ads when the car is approaching a certain area.

While the smart advertising billboard scenario does not have specific requirements in terms of processing and communication delays, the smart traffic light and smart streetlight scenarios may call for very short reaction times to be viable. Totally delegating the logic of the application to the Cloud may turn out not to be an efficient approach, especially while experiencing workload bursts. Much more effective could be an approach that allows the smart car to dynamically discover services and directly consume them without the mediation of the Cloud, in a totally distributed fashion. However, for typical service discovery frameworks, e.g., AllJoyn[3], the underlying technologies may require the discovery phase to be carried out within a single broadcast domain. Packing all services for the Smart City within such a global scope may be incur in scalability issues.

The virtual network instantiation and deallocation functionalities provided by Stack4Things can be very helpful in this regard. In fact, the smart car could be dynamically added to virtual networks of much smaller scope, depending on the city area it is traversing, discovering and consuming only the services that are listening within the broadcast domain associated with that specific area. The part of the application logic that is totally delegated to the Cloud could simply be represented by the automatic addition of the smart car to the specific virtual network(s) associated with the traversed area(s) while the rest of the application logic could be deployed and run on the smart car and other City services-related smart objects, in a Fog computing-oriented style.

In order to do that, Stack4Things can be exploited as follows. The developer designs and implements a set of plugins that she injects into the Stack4Things Cloud and then deploys onto the smart car and the smart objects. The plugins that need to be deployed on the smart car are: A, periodically samples the car position and sends it to the Stack4Things Cloud; B, implements an AllJoyn client for the set of services that the smart car is supposed to discover and consume; C, implements AllJoyn services to be discovered and consumed by the smart car and as such gets deployed on a number of heterogeneous smart objects. Figure 4a depicts this first phase of the use case.

[3]*http://allseenalliance.org*

Once the plugins have been deployed, the developer injects into the Stack4Things Cloud a set of CEP rules that, by analyzing the stream of geolocation samples sent by plugin A on the smart car, detects the events in which the car enters specific areas of interest (see Figure 4b). Depending on its position, addition or removal of the smart car to/from specific virtual networks is thus triggered. As soon as the smart car gets added to a new virtual network, it discovers the services in the area and consumes them according to the logic specified in the plugins (see Figure 4c). At the application layer, the interactions between the smart car and a specific City-provided smart object are direct, without any mediation by the Stack4Things Cloud, which only provides communication services at the network layer.

B. *Quantitative evaluation*

Here we are going to report a quantitative analysis of the above described use case, focusing on the smart traffic lighting example. Let us assume a car approaching a smart traffic light, considering a reasonable cruising speed of 35 km/h for urban traffic and an (estimated) braking distance of about 7 mt: we want to compute the minimal size of the area compatible with requirements deriving from such situation, such as the ability to avoid hard braking and ensure driving safety by flipping the state of the lights (e.g., from red to green) in due time for the driver to react accordingly.

We may therefore compute the mean time for the car to consume the AllJoyn-powered “smart traffic light” services, and this time lapse may be decomposed into a number of phases. Chronologically, first comes the sampling time, i.e., the inverse of the sampling frequency for geolocation coordinates, a constant up to the developer to choose, in our case equal to 1 sec. Then there is the time needed by the CEP to detect that the coordinates fall within the area of interest.

Table I
LATENCY

Phase	**Mean time (sec)**	**Standard deviation**
Position sampling	1	-
CEP detection	1.81	0.14
Network	0.60	0.26
Service	2.48	0.38
Brake	0.72	-
Total	**6.61**	**0.48**

One more latency is related to the time needed to add the car to the virtual network. Last comes the time needed for any AllJoyn services to be discovered by the client running in the car. To take into consideration such braking distance, the derived delay value (0.72 sec) has to be added up.

We performed a set of experiments measuring the previously introduced time intervals, repeating tests a hundred times on a testbed of Arduino[4]-based YUN boards. The mean values and corresponding confidence intervals are reported in Table I. Considering the resulting summation, we may thus derive the size of the area, to be encoded by means of the corresponding parameters (minimum and maximum longitude / latitude) for the CEP. In particular the longest side of the (rectangular) area will be as long as the sum times the aforementioned cruising speed (i.e., 6.61 sec $\times$ 35 km/h$\times 1000$ m/km$\times 1$ h$/3600$ sec $= 64.26$ m), one of its shortest sides being hinged onto the smart light, in our view a remarkable achievement because it enables us to manage urban areas with a high degree of granularity, and thus limits exposure of Smart City services just to potentially interested parties, being involved due to proximity.

V. DISCUSSION AND CONCLUSIONS

In Smart Cities, thousands of smart objects interact among each other implementing complex applications. Fog computing is a new paradigm envisioning such applications to run at the edge of the Internet, ensuring scalability and low latency. In this paper, we presented Stack4Things showing how it can be exploited as a framework for seamless deployment and execution of Fog applications. In fact, Stack4Things provides mechanisms for Cloud-based virtualized networking, contextualization, and complex event processing that we demonstrated to be useful basic functionalities for implementing reconfigurable Smart Cities. Considering a smart mobility scenario, we implemented a use case in which a smart car interacts with Smart City objects to provide the user with a set of geolocalized services. A quantitative analysis demonstrates that our framework allows to effectively manage small areas within a urban region thus providing scalability to fully exploit Fog computing potential. In terms of future work, the authors are considering to investigate mechanisms in support of scalable development and deployment of plugins, such as, e.g., validation of injected code and plugin dependency resolution, respectively.

[4]http://www.arduino.org

REFERENCES

[1] M. Naphade, G. Banavar, C. Harrison, J. Paraszczak, and R. Morris, “Smarter cities and their innovation challenges,” *Computer*, vol. 44, no. 6, pp. 32–39, Jun. 2011.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. ACM, 2012, pp. 13–16.

[3] M. Satyanarayanan, R. Schuster, M. Ebling, G. Fettweis, H. Flinck, K. Joshi, and K. Sabnani, “An open ecosystem for mobile-cloud convergence,” *Communications Magazine, IEEE*, vol. 53, no. 3, pp. 63–70, March 2015.

[4] A. Mukherjee, H. Paul, S. Dey, and A. Banerjee, “Angels for distributed analytics in iot,” in *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, March 2014, pp. 565–570.

[5] M. Habib ur Rehman, C. S. Liew, and T. Y. Wah, “Uniminer: Towards a unified framework for data mining,” in *Information and Communication Technologies (WICT), 2014 Fourth World Congress on*, Dec 2014, pp. 134–139.

[6] J. Preden, J. Kaugerand, E. Suurjaak, S. Astapov, L. Motus, and R. Pahtma, “Data to decision: pushing situational information needs to the edge of the network,” in *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2015 IEEE International Inter-Disciplinary Conference on*, March 2015, pp. 158–164.

[7] S. Distefano, G. Merlino, and A. Puliafito, “Sensing and actuation as a service: A new development for clouds,” in *Network Computing and Applications (NCA), 2012 11th IEEE International Symposium on*, Aug 2012, pp. 272–275.

[8] G. Merlino, D. Bruneo, S. Distefano, F. Longo, and A. Puliafito, “Stack4things: Integrating IoT with OpenStack in a Smart City context,” in *Proceedings of the IEEE First International Workshop on Sensors and Smart Cities*, 2014.

[9] S. Datta, C. Bonnet, and J. Haerri, “Fog computing architecture to enable consumer centric internet of things services,” in *Consumer Electronics (ISCE), 2015 IEEE International Symposium on*, June 2015, pp. 1–2.

[10] H.-L. Truong and S. Dustdar, “Principles for engineering iot cloud systems,” *Cloud Computing, IEEE*, vol. 2, no. 2, pp. 68–76, Mar 2015.

[11] K. Habak, M. Ammar, K. Harras, and E. Zegura, “Femto clouds: Leveraging mobile devices to provide cloud service at the edge,” in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, June 2015, pp. 9–16.

[12] S. Distefano, G. Merlino, and A. Puliafito, “Device-centric sensing: an alternative to data-centric approaches,” *Systems Journal, IEEE*, vol. PP, no. 99, pp. 1–11, Oct 2015.

[13] G. Merlino, D. Bruneo, F. Longo, S. Distefano, and A. Puliafito, “Cloud-based Network Virtualization: An IoT use case,” in *Ad Hoc Networks*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, 2015, vol. 155, pp. 199–210.

[14] “WAMP [URL],” http://wamp.ws.