An SLA-based Approach to Manage Sensor Networks as-a-Service

Valentina Casola*, Alessandra De Benedictis*, Massimiliano Rak†, Giuseppe Aversano‡ and Umberto Villano‡
*Department of Electrical Engineering and Information Technology
University of Naples Federico II
Naples, Italy
Email:{casolav,alessandra.debenedictis}@unina.it
†Second University of Naples
Aversa, Italy
Email: massimiliano.rak@unina2.it
‡Department of Engineering
University of Sannio
Benevento, Italy
Email: {villano,giuseppe.aversano}@unisannio.it

***Abstract*—The integration of sensing infrastructures into the Cloud gives a number of advantages in providing sensor data as a service over the Internet. Many solutions are now available in the literature, and most of them focus on modeling sensor networks as part of the infrastructure to be offered as a service (IaaS), directly managed by means of the Cloud tools that provide resource virtualization. We propose a different approach: sensor networks are modeled as providers that offer their resources to a Cloud application that runs independently from Cloud providers. Being offered as a Service, any user can negotiate with the provider his desired requirements in terms of operational parameters and non-functional features (i.e. security, dependability, ...). In particular, we propose a SLA-based approach for the specification and management of usage term guarantees related to the access and configuration of private sensor networks. To this end, a Cloud Sensing Brokering Platform is designed to illustrate the innovative way to integrate Cloud and Sensor Networks.**

I. Introduction

A sensor network consists of a group of embedded devices provided with a communication infrastructure, intended to monitor and record environmental parameters. Sensor networks are widely adopted in several application domains, including industrial automation, video surveillance, traffic monitoring, air traffic control, disaster management, etc.

In order to promote sensor networks' accessibility and interoperability, the Open Group Consortium has recently defined the Sensor Web Enablement (SWE) Architecture [15], a framework of open standards for exploiting Web-connected sensors and sensor systems. The aim of SWE is that of providing web accessible sensor networks and archived sensor data that can be discovered and accessed using standard protocols and application program interfaces (APIs). By adopting this framework, it is possible to quickly discover sensors having the desired requirements (location, observable parameters, quality, ability to task), obtain information about sensors and observations in a standard encoding, and task sensors, when possible, according to specific needs. These concepts have been taken a step further by the adoption of the Cloud paradigm: several research and industrial efforts have been recently focused on the development of Cloud-based sensing infrastructures, that integrate large-scale sensor networks with sensing applications and Cloud computing infrastructures. The integration of sensors into the Cloud enables users to easily collect, access, process, visualize, archive, share and search large amounts of sensor data from different applications and supports complete sensor data life cycle from data collection to the back-end decision support system.

While several solutions exist to remotely access distributed sensor networks and provide interoperability among different monitoring systems, the usability of real-world sensor networks is still limited. Indeed, there is a lack of general and efficient approaches for dynamically providing guarantees to both users and network owners about the way such networks are used, in terms of regulations on the access to sensed data by multiple external users, and the possibility of partially configuring these networks (e.g. installation of custom applications).

The deployment and management of sensor networks devoted to monitor specific processes or phenomena (e.g. volcanic or seismic activities) in fact, are typically exclusively performed by the sensor network's owner, that usually does not allow external users to freely configure and task them – differently from what happens for most of the existing sensor network testbeds made available by several universities and research centers for the evaluation of protocols and algorithms. The usability of such networks could be maximized by introducing proper mechanisms to dynamically request and obtain guarantees about the usage terms of a target sensor network. Each user could explicitly negotiate, with the network's owner, several parameters such as the exclusivity of usage of the network for a certain interval of time, the features of the monitoring application to install on sensors, or even the physical topology to deploy. On the other side, the network's owner could ask for the fulfillment of some requirements such as the level of authorization of the users or the guarantees on the disclosure of sensed data. Moreover, given a physical location and a set of phenomena of interest, there could be several sensor networks deployed for their monitoring, owned

by different providers with different policies and features. In this scenario, it would be desirable to have the possibility of selecting the provider that best meets the user's requests, for example in terms of flexibility or performance.

The described features can be faced through the adoption of the Cloud computing paradigm applied to sensor networks. As we will outline in the next sections, at the state of the art a lot of solutions exist in this direction, and most of them focus on modeling sensor networks as part of the infrastructure to be offered as a service. For this reason, sensor networks are usually directly managed by means of the Cloud tools that provide resource virtualization. In this paper, we propose a different approach: sensor networks are modeled as providers that offer their resources to a *Cloud application*, which runs independently from Cloud providers (as providers technology and software). Such application acts as a third party, by offering services to both sensor network providers, that share their networks to a larger user community, and customers, that are able to access a large variety of different sensor networks, being able to search among them and negotiate the features they need.

We propose a SLA-based approach for the specification and management of usage term guarantees related to the access and configuration of private sensor networks, where such guarantees are seen from the point of view of both end-users and network owners. A service-level agreement (SLA) is a part of a contract that formally defines the agreement between two or more parties about the service that is to be delivered. In the considered scenario, sensor networks' owners offer to clients a service represented by the access to their network: proper SLAs can be defined including functional and non-functional requirements, regarding both the physical configuration of sensor networks and several general aspects such as desired/provided performance, security and reliability. We introduce a Cloud infrastructure for the negotiation of the service guarantee terms, which aggregates different network providers offering the access to their private sensor networks to clients having specific requirements. Network providers register themselves with a Cloud Sensing Brokering Platform, that will be used as a broker by end-users to find the best matching between her requirements/guarantees and those of each network provider.

The paper is organized as follows: Section II presents some related works about sensors and Cloud, while in Section III we discuss the adopted approach and the services offered by the proposed infrastructure. Section IV presents the architecture of the Cloud Sensing Brokering Platform and, finally, Section V illustrates the whole approach referring to a WSN case study.

II. Related work

The integration of sensing infrastructures into the Cloud gives the advantage of providing sensor data or sensor events as a service over the Internet. In [4], the authors survey some typical applications of Sensor Networks using Cloud computing as backbone. In these applications, the combination of WSNs with Cloud makes it easy to share and analyze real time sensor data on-the-fly, exploiting the virtually unlimited computational and storage capabilities of the Cloud infrastructure. In transport monitoring for example, sensors are used to control traffic lights or detect vehicles and estimate their speed, in order to build a global traffic picture. Data available from sensors is acquired and transmitted for central fusion and processing: this requires storage of data and huge computational cycles, as well as the capability of analysis and prediction of data to generate events. These tasks can be accomplished by means of the integration with the Cloud computing infrastructure. In the military field, Cloud computing may be a solution to the problem of providing a secure infrastructure to protect data collected from military applications, needing a top level security that is not guaranteed using normal Internet connectivity. Finally, in Weather Forecasting, each weather station is equipped with sensors to sense various parameters such as wind speed/direction, relative humidity, temperature (air, water and soil), barometric pressure, precipitation, soil moisture, ambient light (visibility), sky cover and solar radiation. The data collected from these sensors is huge in size and is difficult to maintain using the traditional database approaches. Such data could be processed according to the complicated weather forecast equations, by proper supercomputers accessible within the Cloud infrastructure.

Several research papers and industrial applications have been proposed dealing with sensors and Cloud, identifying two main different approaches. On the one hand, some authors consider sensing and actuation resources in the same way that computing and storage resources are in more traditional Cloud stacks: abstracted, virtualized, and grouped in Clouds [17], [12], [13]. On the other hand, other solutions consider sensors involved in the Cloud exclusively as simple endpoints.

According to the first approach, Sheng et al. proposed in [17] to leverage the Cloud computing model to provide various sensing services using mobile phones, and introduced the concept of *Sensing as a Service* (S^2aaS). In a S^2aaS Cloud, multiple sensing servers can be deployed to handle sensing requests from different locations. When a Cloud user initiates a sensing request through an online form, the request will be forwarded to a sensing server which will then push the request to a subset of mobile phones that are in the area of interest. The sensing task will be fulfilled by these mobile phones, and sensed data will then be collected by the server, stored in the database and returned to the requester. A mobile phone user can be not only a Cloud (service) user who can request sensing services from the Cloud, but also a service provider who fulfills sensing tasks according to sensing requests from other Cloud users.

Mitton et al. introduced in [13] the concept of *Sensing and Actuation as a Service* (SAaaS), and proposed a Cloud of Things (CoT), that provides services by abstracting, virtualizing, and managing things according to the needs and the requirements specified by users, negotiated and agreed to by the parties through specific SLA agreements/procedures.

Yuriyama et al. proposed in [12] the Sensor-Cloud infrastructure, for managing physical sensors on IT infrastructures. The Sensor-Cloud Infrastructure virtualizes a physical sensor as a virtual sensor in the Cloud infrastructure, so that dynamically grouped virtual sensors can be automatic provisioned when the users need them.

The remaining part of the existing work follows the second approach, according to which sensing platforms are only

considered as endpoints for the generation of data, that are then processed by means of the Cloud resources. Indeed, this approach is more suited when considering classical sensor networks, that are actually deployed in many scenarios and are composed of very simple devices, only provided of limited sense and forward capabilities. Moreover, sensor networks have several peculiar features that heavily differentiate them from other systems: firstly, the validity of data generated by a sensor network depends on the physical location of sensor nodes and on the observation time and, secondly, a sensor node cannot be virtualized as other computing resources typically are in a Cloud context, since each node must be physically dedicated to a single task at each time.

The IoTCloud project [11] consists of an open source middleware for Internet of Things (IoT) applications. The IoTCloud is a Cloud based controller for distributed Sensor Grids, which supports an extensible set of sensor-types and large numbers of geographically distributed smart objects. The framework is composed of an IoTCloud Controller for managing the other system components and providing SOAP Web Services for sensor registration, discovery, subscription and control, a Message Broker, that handles the low level details of message routing, sensors and Clients, that subscribe to (consume) sensor data for some application specific purpose. Interoperability is achieved by having sensors and clients interact with the IoTCloud through a set of standards-based SOAP Web Services.

Finally, among industrial applications, MicroStrains SensorCloud^{TM}[2] is a sensor data storage, visualization and remote management platform, that leverages Cloud computing technologies to provide data scalability, rapid visualization, and user programmable analysis. Core SensorCloud provides a virtually unlimited data storage for collecting and preserving long-term sensor data streams, offers time series visualization and graphing tool to allow viewers to navigate through massive amounts of data, and allows users to quickly develop and deploy data processing and analysis applications that live alongside their data in the Cloud. Moreover, it allows users to create custom alerts for monitoring events of interest.

III. Sensor Network SLA-negotiation

In this paper, we propose an SLA-based approach for the specification and management of usage term guarantees related to the access and configuration of private sensor networks offered, through a Cloud infrastructure, by different network providers to multiple end-users. This can be achieved by means of a negotiation process that involves the SLAs defined by each of the interacting parties. Our proposal is motivated by the need, in real-world monitoring systems, of specifying a set of requirements that must be satisfied by both end-users and network providers, in order to come to an agreement on the terms of usage of a specific sensor network.

In order to aggregate multiple network providers to give the users a wider offer, we introduce a Cloud Sensing Brokering Platform (SBP). The SBP is a third party platform that acts both as a broker for the end-users, by helping them in selecting the network providers that best meet their requests, and as a registrar for the network providers, by registering all available monitoring systems and exposing their features to users. Besides managing the accounting phase, the SBP is also in charge of performing the negotiation process among users and network providers on their behalf, while the resulting contract is signed by the two parties. A different approach, that we will not consider in this paper, would see the SBP as the interface between users and providers in all phases of their interaction, with the SBP taking on all the responsibility for the agreements.

It is worth noting that the SBP we propose is a cloud-provider that is independent from the platform, it can be run as a *cloud application* consuming resources of any IaaS cloud provider. This is a clear difference with all the existing solutions, that mix Cloud and sensor solutions, and they are typically offered as a whole Infrastructure as a Service Provider, that own and manage sensors as resources and offer them as a service.

As previously said, the agreement between end-users and network providers about the terms of usage of offered sensor networks is formally defined in a SLA. The SLA Lifecycle as described by the TeleManagement Forum [10] can be split up in six different phases:

1) development of service and service templates,
2) discovery and negotiation of an SLA,
3) service provisioning and deployment,
4) execution of the service,
5) assessment and corrective actions during execution (parallel phase to execution of the service), and
6) termination and decommission of the service.

In this section we illustrate the discovery of the network providers that satisfy users' requirements, and the negotiation with those candidates to reach an agreement (the SLA) on the service requested and ultimately provided. We also refer to the generation of SLA Templates to make requests and to the execution of services.

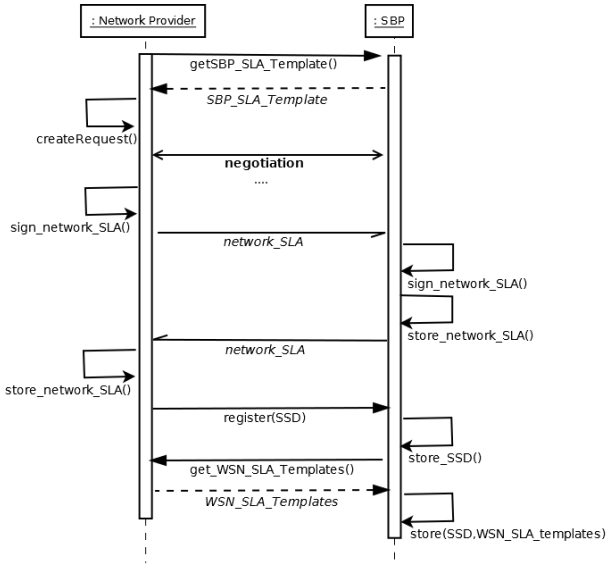

Fig. 1. Network provider registration process

Figures 1 and 2 illustrate the sequence diagrams related respectively to the registration of network providers and users

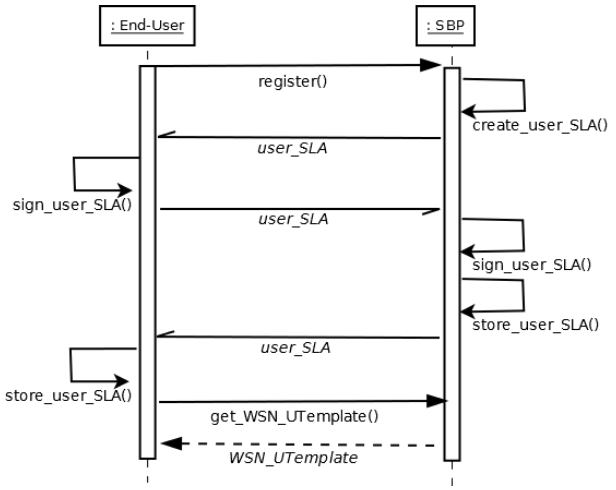

Fig. 2. End-users registration process

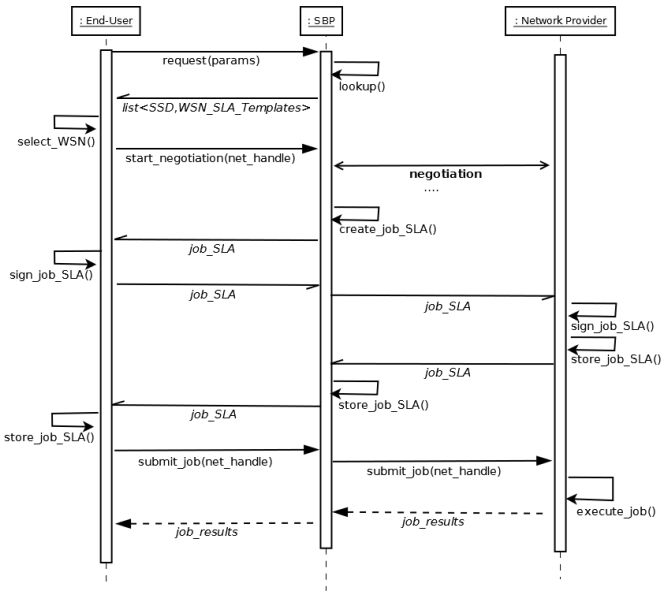

Fig. 3. Job submission process

at the SBP. Each network provider registers its services with the SBP in order to make them available to end-users. Before that, a negotiation process must be carried out between the network provider and the SBP to come to an agreement on the service guarantees provided by the platform to the network provider (e.g. the platform ensures that all communications will be encrypted). Negotiation is performed according to the WS-Agreement specification [5]: the initiator sends a request for an offer (called *Bid*) to the service provider, that sends back an offer to be agreed upon. In order to start negotiation, a network provider retrieves the $SBP\_SLA\_Template$ from the SBP, containing all the terms that can be agreed upon, and builds a request according to its format. At the end of the negotiation, that can include several phases, a $network\_registration\_SLA$ is prepared by the SBP and signed by both parties.

After the agreement has been signed, the network provider can register its services by submitting a Sensing Service Description (SSD), that is a description of the offered services in terms of the features of the exposed sensor network. A service can be offered according to different configurations or quality levels, corresponding to different $SLA\_Templates$ created by the network provider. All templates are retrieved by the SBP, that stores them into a WSN SLA Template Repository.

As for user registration, shown in Figure 2, there is no need to negotiate the services offered by the SBP: the two parties simply sign a $user\_registration\_SLA$ prepared by the SBP, and after that the user retrieves a $SLA\_U\_Template$ (a *union* Template) from the SBP, including all the terms that can be negotiated and have been specified by all the registered WSN providers in their SLA Templates. This template is then used by the user for submitting its service requests.

Figure 3 shows the process of discovery and negotiation of services offered by network providers. When a user wants to search for a network that meets her needs, she will first register with the SBP following the process previously described. After that, the user will build a request containing her own operational requirements about network functional and non functional parameters, by filling the $SLA\_U\_Template$ the has previously retrieved. Upon receiving the user's request, the SBP searches for a match between users' requests and providers' offers and selects potential candidate services among the available configurations described by the SLA templates in its WSN Template Repository. The user selects one of the candidate services and starts a negotiation process, that is actually performed by the SBP on her behalf. The SBP creates a job_SLA specifying all service terms based on the user's request, and asks the end-user and the network provider to sign it. This SLA contains also a description of the actual invocation of the requested service (e.g. sampling frequency, allocation time etc.), and can thus be considered as a job descriptor. After the negotiation, the user can finally access the selected network, by submitting her jobs to the SBP, that in turn forwards them to the network provider. Similarly, job results are sent back to the user by passing through the SBP.

Based on the previous discussion, we can summarize the services offered to network providers and end-users as follows:

- Services offered to network providers
  - GET_SBP_SLA_TEMPLATE: get the SLA_TEMPLATE containing all the possible features that can be negotiated with the SBP
  - REGISTRATION: the network provider registers its services with the SBP by signing a $network\_registration\_SLA$
  - MONITORING: the SBP generates statistics on network usage
- Services offered to end-users
  - REGISTRATION: the user registers at the SBP by signing a $user\_registration\_SLA$
  - GET_U_TEMPLATE: the user retrieves the $SLA\_U\_Template$ containing all the possible features that can be negotiated with the network providers
  - REQUEST: the user submits a service request to the SBP containing her functional and non

functional requirements and obtains a list of candidate services
- SELECT_NET: the user selects a sensor network to connect to from the list of candidate services
- SUBMIT_JOB: the user submits her jobs to the selected network

Moreover, the SBP offers some additional services, such as a support service to create SLAs, a service for users and requests monitoring, logging, alerting on the network status, maintenance requests etc.

IV. Proposed architecture for SLA negotiation and management

As previously discussed, our goal is to design a framework for the aggregation of different sensor providers that offer the access to their private sensor networks to external users having specific requirements. This is achieved by means of the introduction of proper SLAs, defining the usage terms guarantees related to such networks. In this section, we present the architecture of the proposed Cloud Sensing Brokering Platform, the Cloud application that has been developed for the management and negotiation of SLAs. As such architecture is based upon the mOSAIC framework, we are going to give an overview of it in the following subsection, before discussing the details about the architecture components in Section IV-B.

A. mOSAIC: Development of Cloud Applications

mOSAIC [14], [9] is a framework that provides an API to develop cloud applications, which are thereafter executed in a leased environment provisioned and controlled by the mOSAIC run-time. Hence, the target user for the mOSAIC solution is the application developer (*mOSAIC user*). In mOSAIC, a cloud application is structured as a set of components running on cloud resources (i.e., on resources leased by a cloud provider) and able to communicate with each other. Cloud applications are often provided in the form of Software-as-a-Service, and can also be accessed/used by other users than the mOSAIC developer (i.e., by *final users*). In this case, the mOSAIC user acts as service provider for final users.

A mOSAIC application is built up as a collection of interconnected *mOSAIC components*. Components may be (i) core components, i.e., predefined helper tools offered by the mOSAIC platform for performing common tasks, (ii) COTS (commercial off-the-shelf) solutions embedded in a mOSAIC component, or (iii) *cloudlets*.

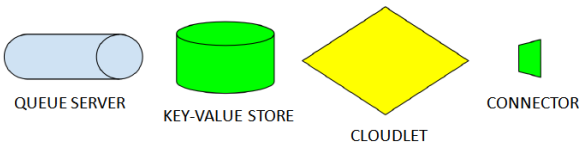

Fig. 4. mOSAIC components

Basic mOSAIC components, and the symbols used to represent them, are depicted in Figure 4. Queues and Key-Value Stores provide support for inter-component communication and storage of data. *Queue servers* in particular are based on standard protocols (AMQP) and technologies (RabbitMQ) and allow queue-based communications among other components: each queue server is able to manage one or more queues and mOSAIC components can be registered both as publishers and consumers upon such queues. *Key-value Stores* instead, use standard technologies (such as Riak) to allow non-relational storage and retrieval of data from other components.
Cloudlets are the programmable components that encapsulate the core logic of the specific application. The mOSAIC API, through which cloudlets are described, promotes an event-driven programming style, which results in an asynchronous execution model that allows higher scalability. Cloudlets are executed in special containers and are able to self-scale and to interact with any kind of cloud resource. Connection between cloudlets and other components is accomplished by means of proper *connectors*.

A cloud application is described as a whole in a file named *Application Descriptor*, which lists all the application components (cloudlets), the cloud resources (queues and key-value stores) and the details of their interconnections. A mOSAIC developer has the role both of developing new components and of writing application descriptors that connect them. All the mOSAIC components run on a dedicated virtual machine, named mOS (mOSAIC Operating System), which is based on a minimal Linux distribution. The mOS is enriched with a special mOSAIC component, the *Platform Manager*, which makes it possible to manage a set of virtual machines hosting the mOS as a virtual cluster, on which the mOSAIC components are independently managed. It is possible to increase or to decrease the number of virtual machines dedicated to the mOSAIC Application, which will scale in and out automatically.

B. The Sensing Brokering Platform architecture

The overall architecture of the proposed framework is presented in Figure 5: the SBP is made of different interconnected components that, together, implement the services described in the previous section. The *Negotiation* cloudlet is the core component of the architecture, aimed at managing the whole process of agreement negotiation and job submission on behalf of users. It collects users' requests and providers' offers through the micro-http gateway *mhttpgw*, and properly invokes the other components to perform the specific requested tasks (i.e. registration, SLA management, network selection etc.).

As shown in the figure, several key-value store components have been deployed, in order to provide storage facilities to the stateless cloudlets. The *NP_SLA KV*, *User_SLA KV* and *Job_SLA KV* components are used to store agreements related respectively to the registration of network providers, the registration of users and the negotiation about job submissions. These agreements are stored along with an information about their current state and can be accessed in an associative way. References to the correspondence between keys and agreements' IDs are contained in additional key-value store servers (respectively, the *NP_keys*, *User_keys* and *Job_keys* components) for search purposes.

The state of each agreement is updated by means of proper *Change_State* messages, generated by the *Negotiation* cloudlet and sent to the *NP_SLA_store*, *User_SLA_store* and *Job_SLA_store* cloudlets during the negotiation process.

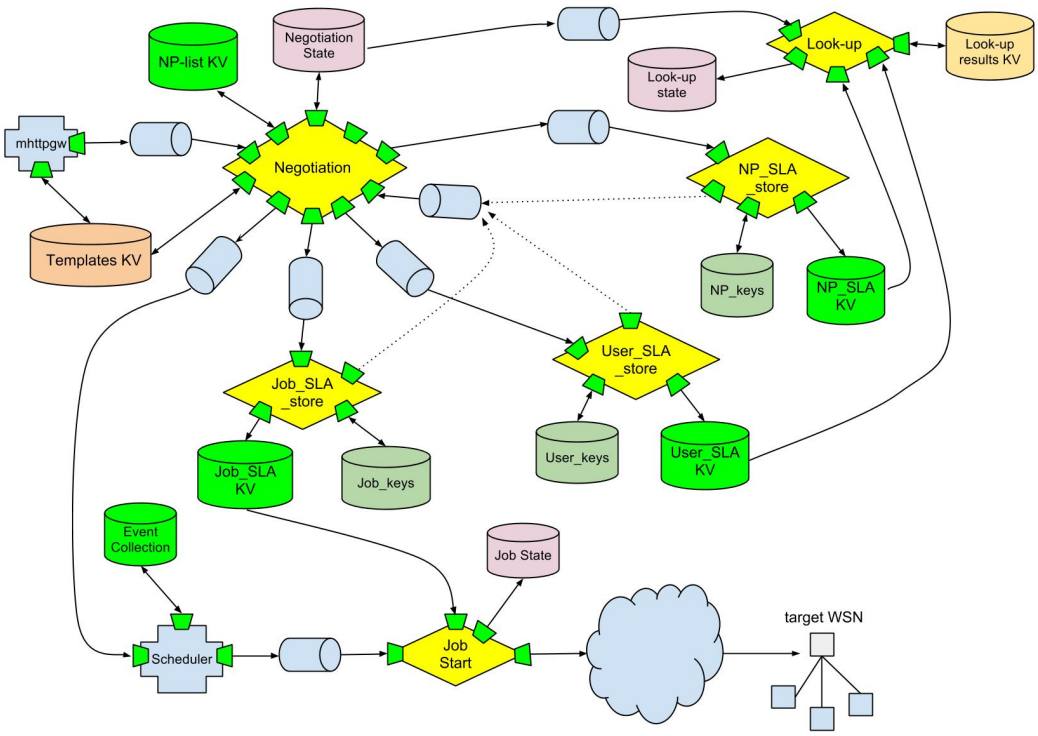

Fig. 5. SBP architecture

The *Templates KV* key-value store contains all SLA templates involved in the different negotiations, that is those used during the registration phase and the WSN Templates generated by network providers to describe the offered services.

During the registration phase, network providers retrieve the $SBP\_SLA\_Template$, stored in the *Templates KV* key-value store, and start negotiation with the SBP. At the end of such negotiation, a $network\_SLA$ is signed by both parties and stored by the *NP_SLA_store* cloudlet into the *NP_SLA KV*. Also, the *Negotiation* cloudlet adds an entry to its *NP-list KV* to keep track of the new registered provider. Meantime, the state of the negotiation process is updated within the *Negotiation State* server, that can be accessed by the negotiation initiator to check for its completion. Once the registration is complete, the network provider submits its WSN templates to the SBP, that will store them into the *Templates KV* server.

The user registration, as previously said, does not include a real negotiation with the SBP: the user simply retrieves a $user\_SLA$ template from the SBP (contained in the *Templates KV* server) and signs it, and this agreement is then stored into the *User_SLA KV*. Afterwards, the user retrieves the $SLA\_U\_Template$, that is generated by the *Negotiation* cloudlet by merging all the parameters that have been specified by registered network providers in their templates.

When the user wants to submit a job to one of the available sensor networks, she prepares a request according to the *SLA_U_Template* and sends it to the SBP through the *mhttpgw* interface. This request is received by the *Negotiation* cloudlet, that retrieves all network IDs contained in its *NP-list KV* and passes them, together with the user's ID and the request itself, to the *Look-up* cloudlet.

The *Look-up* cloudlet accesses the *NP_SLA KV* and the *User_SLA KV* to retrieve the user SLA and all network providers' SLAs, in order to search for the best match between the request and all the offers. The state of the look-up process is stored and updated in the *Look-up State* server, to give a feedback to the user. Once the process is completed, the resulting matching SLAs are stored into the *Look-up results KV* key-value store and retrieved by the user, that will select one of the candidates. After that, the user starts the negotiation process, that is handled by the *Negotiation* cloudlet. The resulting *job_SLA* is signed and stored by the *Job_SLA_store* cloudlet into the *Job_SLA KV*.

At the end of negotiation, an event is generated by the Negotiation cloudlet toward the *Scheduler* component, that manages the scheduling of jobs over time. All job events are stored in an *Event Collection* key-value store, sorted based on the job delays (corresponding to the job activation time). When a job is scheduled, the *Job Start* cloudlet will take care of it by forwarding job requests to the selected network provider. In order to give a feedbacjk to the user, the state of the job is stored and updated in the *Job State* server.

V. CASE STUDY

In this section, we present a case study aimed at demonstrating the possible parameters that can be negotiated about the usage of a sensor network offered by a network provider. We will refer to some sensor networks that we deployed in previous experiments to protect critical infrastructures [7] and in our laboratories [6], [8]. We have a testbed with 8 sensors, grouped by 2 networks of 4 sensors each: the first network is made of TelosB nodes [3] and measures temperature, humidity and GPS coordinates, while the second network is made of MicaZ nodes [1] and measures acceleration and GPS coordinates. Note that micaZ boards have usually better performance than TelosB. For each network, a gateway board linked to a PC communicates via a WiFi connection with the control room.

Each network can implement different security mechanisms to protect the communication among nodes, depending on the required security level and available resources. Sensor providers can provide two different cryptosystems based respectively on Elliptic Curve Cryptography (WM-ECC libraries) and Identity-based cryptographic techniques (TinyPairing libraries). WM-ECC [16] provides support for all the ECC operations and can be used to implement key agreement and digital signature algorithms, that can be adopted to authenticate packets in the sensor network. We used the WM-ECC library in a previous work [6] to implement a hybrid cryptosystem based on a public key function for ensuring authentication of the base station, and on a key agreement protocol for establishing a symmetric key, to be used for encryption/decryption of data packets sent by the motes. TinyPairing [18] is an open-source pairing-based cryptographic library for wireless sensors, providing an interesting solution to the key management problem, that still represents an open issue in WSN security research.

Each network provider creates and submits to the SBP several SLA Templates that correspond to the different offered services. Service offers advertised in SLA templates typically refer to non-functional requirements, related for example to the provided security and performance levels. In our case, even some functional requirements could be subject to negotiation, such as for example the required sampling frequency or the time interval in which the network should be assigned to the user. Following the proposed approach, the two network providers register at the SBP, providing the different templates. Registration results in signing SLAs, represented in the WS-Agreement format, containing the network description and rules about data acquisition (such as the availability period). For the sake of brevity, we report a brief textual description of the SLA content, in terms of the included parameters, instead of the (verbose) XML code:

- NET1-SLA_template1.*SLA11*: **SECURITY** {wm-ecc}, **JOB** {minimum sample period = 100 ms, telosb hw, temperature and pressure sensors, available within 7 days}

- NET1-SLA_template2.*SLA12*: **SECURITY** {no security}, **JOB** {minimum sample period = 20 ms, telosB hw, temperature and pressure sensors, available within 7 days}

- NET2-SLA_template1.*SLA21*: **SECURITY** {tinypairing}, **JOB** {minimum sample period = 30 ms, micaZ hw, accelx and accely sensors, available within 15 days}

- NET2-SLA_template2.*SLA22*: **SECURITY** {wm-ecc}, **JOB** {minimum sample period = 15 ms, micaZ hw, accelx and accely sensors, available within 15 days}

Assume a user wants to submit a request containing the following operative/functional requirements and security requirements:

***SECURITY*{tinypairing} *JOB*{sample period=50ms, accelx and accely sensors, start monitoring tomorrow at 12:00 for 3 days}.**

The lookup operation performed by the SBP does not find any matching service, as no one of the available templates fulfills the user's requests: the request is then rejected, and the negotiation process is re-iterated. Assume that later on the user provides a new request, containing the following requirements:

***SECURITY*{tinypairing} *JOB*{sample period=50ms, accelx and accely sensors, start monitoring start monitoring within 15 days and for 3 days}.**

The request is now compliant to one of the provided services, and the negotiator identifies NET2 as the target network the user will send its jobs to. At the end of such negotiation process, a new SLA will be signed (the *job_SLA*), which summarizes the features the user has requested and accepted and locks the resources on the network provider in order to grant the user that his experiments will be correctly carried on.

The code in Listing 1 illustrates the Ws-Agreement SLA signed. Note how in the different sections the global informations are contained, like the expiration date, the target network Provider and the template adopted for the job.

Listing 1. The Job Service Level Agreement (pieces)

```
[..]
    <wsag:Name>SensCloud Provider SLA</
        wsag:Name>
      <wsag:Context>
          <wsag:AgreementInitiator>End User</
              wsag:AgreementInitiator>
          <wsag:AgreementResponder>
              Cloudplatform</
              wsag:AgreementResponder>
          <wsag:ServiceProvider>NET2 Sensor
              Provider </wsag:ServiceProvider>
        <wsag:ExpirationTime>19/7/2013</
            wsag:ExpirationTime>
        <wsag:TemplateId>JobSLA-NET2</
            wsag:TemplateId>
      </wsag:Context>
[...]
  <AppConfiguration>
  [...]
    <security>
      <technique>timypairing</technique>
    </security>
  [...]
    <requiredObservations> //App configuration
        parameters
        <observation>
```

```
        <PhysicalQuantity>accelx</
            PhysicalQuantity>
            <code>obsv1</code>  //SensorML
                coded value
            <sampleFrequency>50ms</
                sampleFrequency>
            <observationTime>3 days</
                observationTime>
        </observation>
      <observation>
        <PhysicalQuantity>accely</
            PhysicalQuantity>
            <code>obsv1</code>  //SensorML
                coded value
            <sampleFrequency>50ms</
                sampleFrequency>
            <observationTime>3 days</
                observationTime>
        </observation>
[...]
```

The negotiated parameters are contained in the *AppConfiguration* section. They are not expressed by means of Ws-Agreement tags, but they are encapsulated in a *ServiceDescriptionTerm* (not reported in the listing for brevity's sake) that contains a custom description derived from the SensorML and Observation&Measurements languages [15], describing the structure of the sensor network and the operational features. Data reported in such description terms enable the Sensor Provider to start the application on the network without additional effort from user.
To conclude, the proposed prototype showed the feasibility of the approach and proposes an innovative way to integrate Cloud and Sensor Networks, maintaining the grants that are typical of hardware-based systems, like sensor networks are, even involving the Cloud, that typically introduces additional software layers that make the system less controllable.

VI. Conclusions and Future Work

In this paper we proposed a Cloud Sensing Brokering Platform dedicated to management of Cloud sensors, which runs as a *Cloud Application* consuming resources (i.e. virtual machines and storages) from different Cloud providers. Such platform is able to dynamically manage and offer to multiple customers different sensor networks made available by existing Sensor Providers. These providers own dedicated infrastructures and are interested in offering them to external users, under specific service guarantees.

The platform was implemented using an innovative Cloud development paradigm, resulting from the mOSAIC Project. The proposed platform has to face contradictory needs from customers, typically researchers with clear constraints on how their measurement campaign should be carried on, and providers, owners of sensor networks that aim at offering their sensor infrastructure, but have fixed constraints on how they can be shared or deployed.

In order to cope with these issues our platform adopts a SLA-based approach: all interactions among customers, network providers ad the Cloud Brokering Platform take place through negotiation of SLAs, reporting the individual needs of each different actor. The platform uses signed SLAs even to represent the single jobs that reserve the sensor networks for the users, offering a fine grained control over the status of the activities.

In future work we aim at improving the platform features, with a more powerful user management, explicit management of security both for controlling customers access and for the security of data produced. Moreover, we have the plan of performing a detailed analysis on the performance of the proposed platform, outlining possible bottlenecks and performance issues.

References

[1] Micaz datasheet.

[2] Microstrains sensorcloud™.

[3] Telosb datasheet.

[4] A. Alamri, W. S. Ansari, M. M. Hassan, M. S. Hossain, A. Alelaiwi, and M. A. Hossain. A survey on sensor-cloud: Architecture, applications, and approaches. *International Journal of Distributed Sensor Networks*, 2013, 2013.

[5] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement).

[6] V. Casola, A. De Benedictis, A. Drago, and N. Mazzocca. Analysis and comparison of security protocols in wireless sensor networks. In *Reliable Distributed Systems Workshops (SRDSW), 2011 30th IEEE Symposium on*, pages 52–56. IEEE, 2011.

[7] V. Casola, A. De Benedictis, A. Drago, and N. Mazzocca. SeNsiM-SEC: secure sensor networks integration to monitor rail freight transport. *In International Journal of System of System Engineering*, 2013.

[8] V. Casola, A. De Benedictis, A. Mazzeo, and N. Mazzocca. Sensim-sec: security in heterogeneous sensor networks. In *Network and Information Systems Security (SAR-SSI), 2011 Conference on*, pages 1–8. IEEE, 2011.

[9] C. Craciun, M. Neagul, I. Lazcanotegui, M. Rak, and D. Petcu. Building an Interoperability API for Sky Computing. In *The Second International Workshop on Cloud Computing Interoperability and Services*, pages 405–406. IEEE, 2011.

[10] TeleManagement Forum. 2005.

[11] G.C. Fox, S. Kamburugamuve, and R.D. Hartman. Architecture and measured characteristics of a cloud based internet of things. In *Collaboration Technologies and Systems (CTS), 2012 International Conference on*, pages 6–12, 2012.

[12] Y. Madoka and K. Takayuki. Sensor-cloud infrastructure - physical sensor management with virtualized sensors on cloud computing. *2012 15th International Conference on Network-Based Information Systems*, 0:1–8, 2010.

[13] N. Mitton, S. Papavassiliou, A. Puliafito, and K. Trivedi. Combining cloud and sensors in a smart city environment. *EURASIP Journal on Wireless Communications and Networking*, 2012(1):247, 2012.

[14] D. Petcu, C. Craciun, and M. Rak. Towards a Cross Platform Cloud API - Components for Cloud Federation. In *CLOSER*, pages 166–169, 2011.

[15] C. Reed, M. Botts, J. Davidson, and G. Percivall. Ogc sensor web enablement: overview and high level architecture. In *Autotestcon, 2007 IEEE*, pages 372–380, 2007.

[16] H. Wang, B. Sheng, C. Tan, and Q. Li. WM-ECC: An elliptic curve cryptography suite on sensor motes. Technical Report WMCS-2007-11, College of William and Mary, October 2007.

[17] S. Xiang, X. Xuejie, T. Jian, and X. Guoliang. Sensing as a service: A cloud computing system for mobile phone sensing. In *Sensors, 2012 IEEE*, pages 1–4, 2012.

[18] X. Xiong, D. S. Wong, and Deng. X. TinyPairing: A fast and lightweight pairing-based cryptographic library for wireless sensor networks. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2010)*, April 2010.