CHAPTER

2 OPEN SOURCE SEMANTIC WEB INFRASTRUCTURE FOR MANAGING IoT RESOURCES IN THE CLOUD

N. Kefalakis, S. Petris, C. Georgoulis, J. Soldatos
Athens Information Technology, Marousi, Greece

2.1 INTRODUCTION

Cloud computing and Internet of Things (IoT) are nowadays two of the most prominent and popular ICT paradigms that are expected to shape the next era of computing. The cloud computing paradigm [1] realizes and promotes the delivery of hardware and software resources over the Internet, according to an on-demand utility-based model. Depending on the type of computing resources delivered via the cloud, cloud services take different forms, such as Infrastructure as a service (IaaS), Platform as a service (PaaS), Software as a service (SaaS), Storage as a service (STaaS), and more. These services hold to promise to deliver increased reliability, security, high availability, and improved QoS at an overall lower total cost of ownership. At the same time, the IoT paradigm relies on the identification and use of a large number of heterogeneous physical and virtual objects (ie, both physical and virtual representations), which are connected to the Internet [2]. IoT enables the communication between different objects, as well as the in-context invocation of their capabilities (services) toward added-value applications. Early IoT applications are based on Radio Frequency Identification (RFID) and Wireless Sensor Network (WSN) technologies, and deliver tangible benefits in several areas, including manufacturing, logistics, trade, retail, and green/sustainable applications, as well as in other sectors.

Since the early instantiations and implementations of both technologies, it has become apparent that their convergence could lead to a range of multiplicative benefits. Most IoT applications entail a large number of heterogeneous geographically distributed sensors. As a result, they need to handle numerous sensor streams, and could therefore directly benefit from the immense distributed storage capacities of cloud computing infrastructures. Furthermore, cloud infrastructures could boost the computational capacities of IoT applications, given that several multisensor applications need to perform complex processing that is subject to timing and other QoS constraints. Also, a great deal of IoT services (eg, large-scale sensing experiments and smart-city applications) could benefit from a utility-based delivery paradigm, which emphasizes the on-demand establishment and delivery of IoT applications over a cloud-based infrastructure.

2.2 BACKGROUND/RELATED WORK

The proclaimed benefits of the IoT/cloud convergence have (early on) given rise to research efforts that attempted to integrate multisensory services into cloud computing infrastructures. Early efforts have focused on the development of pervasive (sensor-based) grid-computing infrastructures [3,4], which emphasized modeling sensors and their data as a resource, and, accordingly, enabling real-time access, sharing, and storage of sensor data [5]. Sensor Grids have been used for a number of pervasive computing applications, notably community-sensing applications such as meteorology [6]. With the advent of cloud computing, the convergence of the cloud computing with WSN infrastructures has been attempted, as an extension of the sensor grid concept in the scope of on-demand elastic cloud-based environments. The convergence of cloud computing with WSN aimed at compromising the radically different and conflicting properties of the two (ie, IoT and cloud) technologies [7]. In particular, sensor networks are location-specific, resource constrained, expensive (in terms of development/deployment cost), and generally inflexible in terms of resource access and availability. On the contrary, cloud-based infrastructures are location-independent and provide a wealth of inexpensive resources, as well as rapid elasticity [7]. Sensor clouds come to bridge these differences and endow WSN with some cloud properties. Other issues are associated with the energy efficiency and the proper handling of service-level agreements [8]. Most recent research initiatives are focusing on real-life implementation of sensor clouds, including open source implementations [9,10].

In addition to research efforts toward sensor-clouds, there are also a large number of commercial online cloud-like infrastructures, which enable end users to attach their devices on the cloud, while also enabling the development of applications that use those devices and the relevant sensor streams. Characteristic examples of such commercial systems include Xively (www.xively.com), ThingsSpeak (www.thingspeak.com), and Sensor-Cloud (www.sensor-cloud.com). These systems provide tools for application development, but offer very poor semantics and no readily available capabilities for utility-based delivery. There are also a number of other projects which have been using cloud infrastructures as a medium for machine-to-machine (M2M) interactions [11], however, without adapting the cloud infrastructure to the needs of the IoT delivery.

Although the previously mentioned projects explore the IoT/cloud integration, they address only a limited number of the issues that surround the IoT/cloud convergence. Specifically, their approach is mostly oriented toward interconnecting sensor streams and IoT services with existing cloud middleware, rather than building a converged cloud/IoT middleware infrastructure that could allow IoT services to fully leverage the capabilities of the cloud. Indeed, by streaming sensor data into the cloud, state-of-the-art projects take advantage of the elasticity and the storage capacity of the cloud in the scope of IoT applications. However, this streaming is not complemented by appropriate resource management mechanisms, which could optimize the usage of cloud resources by IoT applications. Note that efficient resource management is extremely important, given the vast amount of data that could be generated by IoT applications, which could result in high costs for cloud storage. Furthermore, the previously listed IoT cloud platforms feature very poor semantics in terms of the sensor/data streams that they manage, since they only manage minimal metadata that refer to the data streams. This lack of semantics is a serious setback for implementing effective resource management mechanisms, by identifying which sensors and/or data are required in the scope of specific IoT applications. At the same time, the lack of metadata prevents the dynamic selection of data streams and their data in the scope of IoT applications, thereby limiting the flexibility associated with the rapid reuse and repurposing of sensor/data streams across multiple applications. This rapid reuse and repurposing of data streams within the cloud could provide a sound basis for the cost-effective development and delivery of multiple IoT applications/services over the cloud infrastructure. Therefore, the ability to flexibly reuse and repurpose data streams stemming from the same sensors across multiple applications holds the promise to significantly reduce the Total Cost of Ownership (TCO) of the IoT services.

In order to alleviate the resource management issues, the cloud infrastructure needs to keep track of the resources that are consumed/used by the various IoT services. The tracking of these resources is a prerequisite for implementing resource optimization techniques at both the cloud (eg, caching mechanisms) and the sensors/IoT (eg, data streaming according to application needs) levels. This is because the various optimization strategies need to access information about the metadata of the sensors and their data (eg, location, orientation, timestamps, measurement units, reliability, accuracy, cost, data frequency). Furthermore, the richness of the metadata is a factor that could drive the sophistication and efficiency of the resource management schemes. A prominent way of keeping track of the IoT resources in the cloud is the scheduling of IoT services. Scheduling refers to the process of regulating how IoT services access the different resources of the IoT/cloud environment. It implies knowledge about how the various IoT services use the various cloud and sensor resources. The distinction between sensor and cloud resources is required, given that the various sensors are typically owned/managed by different administrative entities from the cloud provider. Although the scheduling concept is straightforward, its implementation is challenging, (mainly) given the volatility of the IoT environments, where sensors join and leave dynamically, at the same time as IoT services are being created and/or destroyed at fine time-scales.

In this chapter we introduce a novel architecture for IoT/cloud convergence, which alleviates several of the limitations of state-of-the-art infrastructures, notably the limitations that are associated with their poor semantics and their inability to support sophisticated resource management mechanisms. The novel characteristics of the introduced architecture are the integration of rich metadata (semantics) concerning the sensors and the data streams, as well as the provision of support for scheduling IoT services in the cloud. In terms of metadata integration, the architecture supports semantic web technologies and standards, including standardized ontologies for describing internet-connected objects and their data streams. In terms of scheduling mechanisms, the architecture provides the means for dynamically establishing IoT services in a way that reserves and keeps track of the resources that they require. Resource reservations are supported at both the (global) level of the cloud infrastructure and at the (local) level of individual sensor deployments. The introduced architecture aims at serving as a blueprint, for rapidly implementing and integrating IoT/cloud solutions. To this end, an open source implementation of its main components/modules is provided as part of the FP7 OpenIoT Project (www.openiot.eu), which is cofunded by the European Commission. In this blueprint direction we also present its use for instantiating and deploying sample IoT solutions. Furthermore, we illustrate how the scheduling process and the semantically rich metadata of the sensors can be used, in order to implement nontrivial resource management mechanisms. Note that the architecture and the modules that are presented in this chapter have been implemented as part of the OpenIoT Open Source Project (https://github.com/OpenIotOrg/openiot).

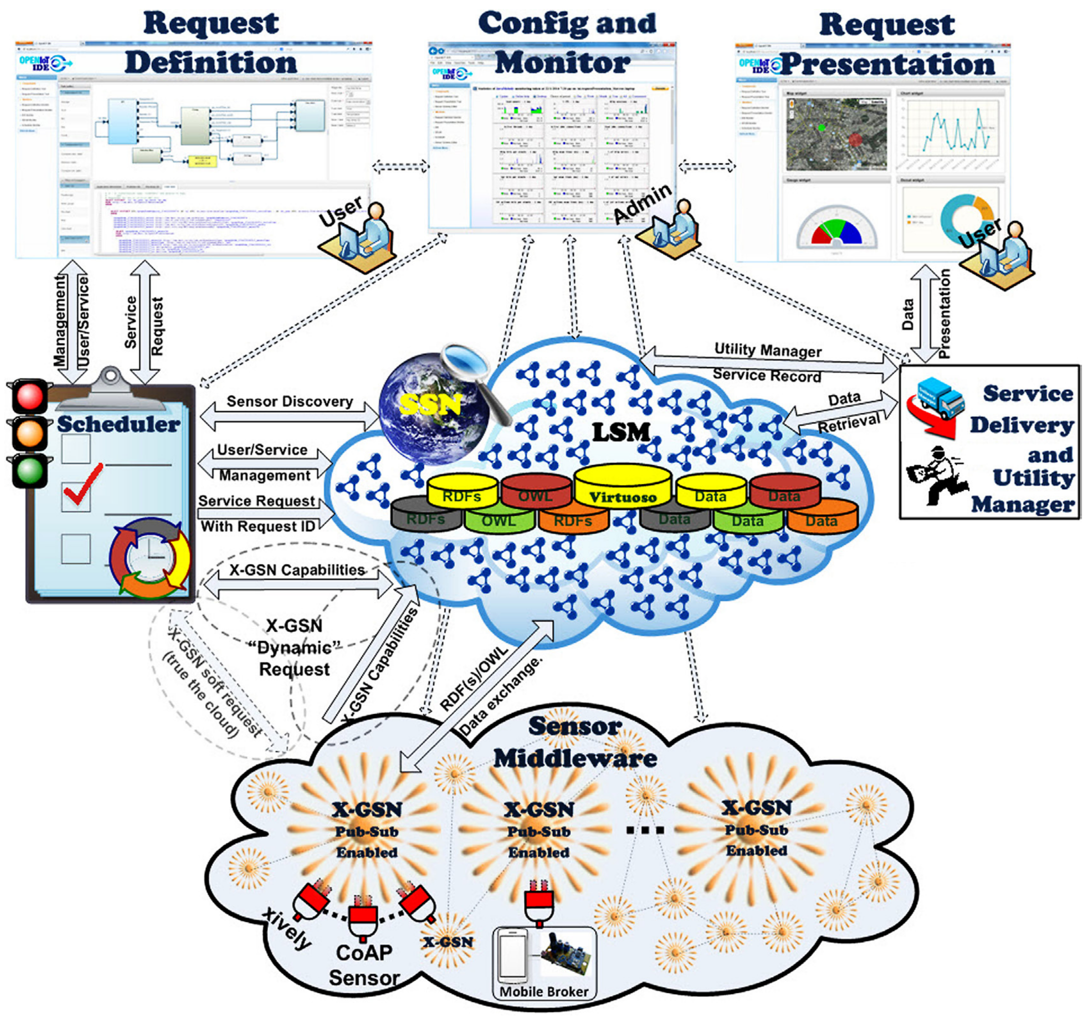


FIGURE 2.1 OpenIoT Architecture for IoT/Cloud Convergence

2.3 OPENIoT ARCHITECTURE FOR IoT/CLOUD CONVERGENCE

Our approach for converging IoT and cloud computing is reflected in the OpenIoT architecture, which is depicted in Fig. 2.1. The figure illustrates the main elements of the OpenIoT software architecture along with their interactions and functionalities, in particular:

- *The Sensor Middleware*, which collects, filters, and combines data streams stemming from virtual sensors (eg, signal-processing algorithms, information- fusion algorithms, and social-media data streams) or physical-sensing devices (such as temperature sensors, humidity sensors, and weather stations). This middleware acts as a hub between the OpenIoT platform and the physical world, as it enables the access to information stemming from the real world. Furthermore, it facilitates the interface to a variety of physical and virtual sensors, such as IETF-COAP-compliant sensors (ie, sensors providing RESTful interfaces), data streams from other IoT platforms (such as https://xively.com), and social networks (such as Twitter). Among the main characteristics of the sensor middleware is its ability to stream sensor data in the cloud, according to semantic format (ie, ontology). The Sensor Middleware is deployed on the basis of one or more distributed instances (nodes), which may belong to different administrative entities. The prototype implementation of the OpenIoT platform uses the GSN middleware [12]. However, other sensor middleware platforms (such as those reviewed in Ref. [13]) could also be used in alternative implementations and deployments of the introduced architecture.

- *The Cloud Computing Infrastructure*, which enables the storage of data streams stemming from the sensor middleware, thereby acting as a cloud database. The cloud infrastructure also stores metadata for the various services, as part of the scheduling process, which is outlined in the next section. In addition to data streams and metadata, computational (software) components of the platform could also be deployed in the cloud in order to benefit from its elasticity, scalability, and performance characteristics. Note that the cloud infrastructure could be either a public infrastructure [such as the Amazon Elastic Compute Cloud (EC2)] or a private infrastructure (such as a private cloud deployed, based on Open Stack). The cloud infrastructure can be characterized as a sensor cloud, given that it primarily supports storage and management of sensor data-streams (and of their metadata).
- *The Directory Service*, which stores information about all the sensors that are available in the OpenIoT platform. It also provides the means (ie, services) for registering sensors with the directory, as well as for the look-up (ie, discovery) of sensors. The IoT/cloud architecture specifies the use of semantically annotated descriptions of sensors as part of its directory service. The OpenIoT open source implementation is based on an enhanced version of the W3C SSN ontology [14]. As a result of this implementation technology, semantic Web techniques (eg, SPARQL and RDF) and ontology management systems (eg, Virtuoso) are used for querying the directory service. Furthermore, the exploitation of semantically annotated sensors enables the integration of data streams within the Linked Data Cloud, thereby empowering Linked Sensor Data. Note that other alternative implementations of the directory services (eg, based on publish/subscribe techniques) are also possible. The Directory Service is deployed within the cloud infrastructure, thereby providing the means for accessing sensor data and metadata residing in the cloud.
- *The Global Scheduler*, which processes all the requests for on-demand deployment of services, and ensures their proper access to the resources (eg, data streams) that they require. This component undertakes the task of parsing the service request, and, accordingly, discovering the sensors that can contribute to its fulfillment. It also selects the resources, that is, sensors that will support the service deployment, while also performing the relevant reservations of resources. This component enables the scheduling of all IoT services, as outlined in the following section.
- *The Local Scheduler component*, which is executed at the level of the Sensor Middleware, and ensures the optimized access to the resources managed by sensor middleware instances (ie, GSN nodes in the case of the OpenIoT implementation). Whereas the Global Scheduler regulates the access to the resources of the OpenIoT platform (notably the data streams residing in the cloud), its local counterpart regulates the access and use of the data streams at the lower level of the Sensor Middleware.

- *The Service Delivery and Utility Manager*, which performs a dual role. On the one hand, it combines the data streams as indicated by service workflows within the OpenIoT system, in order to deliver the requested service. To this end, this component makes use of the service description and the resources identified and reserved by the (Global) Scheduler component. On the other hand, this component acts as a service-metering facility, which keeps track of utility metrics for each individual service. This metering functionality is accordingly used to drive functionalities such as accounting, billing, and utility-driven resource optimization. Such functionalities are essential in the scope of a utility (pay-as-you-go) computing paradigm.
- *The Request Definition tool*, which enables the specification of service requests to the OpenIoT platform. It comprises a set of services for specifying and formulating such requests, while also submitting them to the Global Scheduler. This tool features a Graphical User Interface (GUI).
- *The Request Presentation component*, which is in charge of the visualization of the outputs of an IoT service. This component selects mashups from an appropriate library in order to facilitate service presentation. Service integrators and solution providers have the option to enhance or override the functionality of this component toward providing a presentation layer pertaining to their solution.
- *The Configuration and Monitoring component*, which enables management and configuration functionalities over the sensors, and the IoT services that are deployed within the platform. This component is also supported by a GUI.

Fig. 2.1 does not specify implementation technologies associated with the various components, thus providing an abstract presentation of the functional elements of the architecture. OpenIoT is, however, implemented on the basis of specific implementation technologies [such as GSN for the sensor middleware, W3C SSN for the directory service, and JSF (Java Server Faces) libraries (such as Primefaces)]. Alternative implementations based on alternate technologies are however possible. IoT solution providers and/or service integrators could adopt this architecture as a baseline for providing their own implementation, which may use only part of the open source components and technologies of the OpenIoT platform implementation, which is currently available at: https://github.com/OpenIotOrg/openiot.

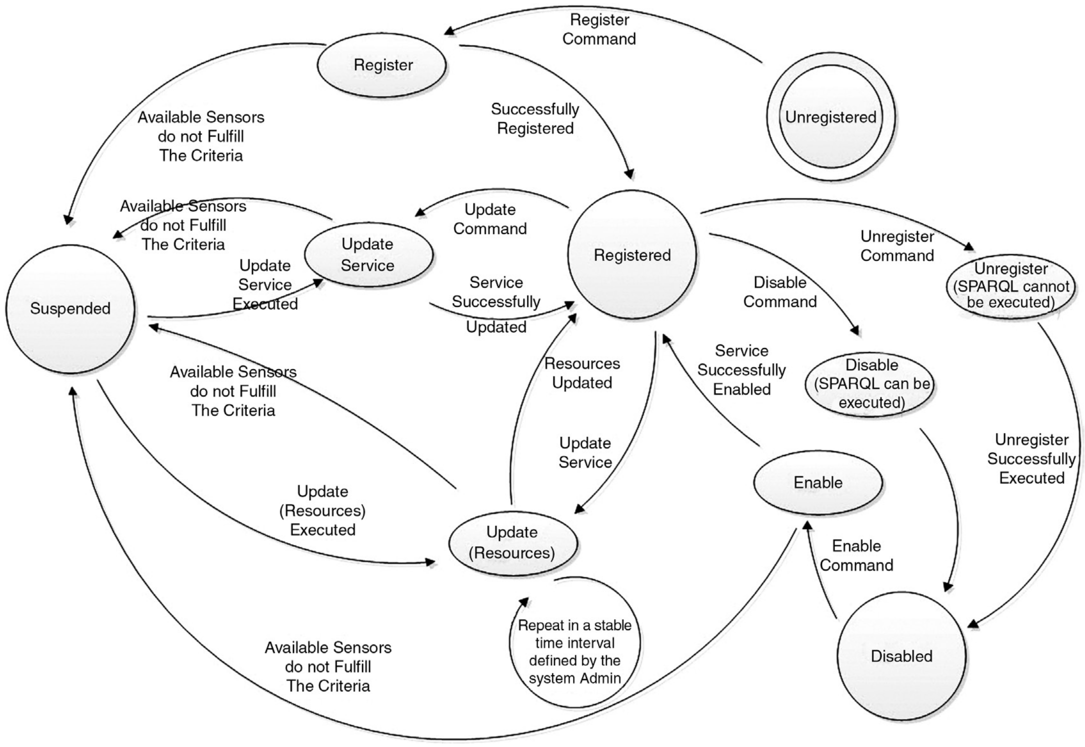
The delivery of IoT services through the platform relies on data collected and streamed into the cloud through the (GSN) sensor middleware. Given the existence of multiple data streams within the cloud, a typical workflow associated with the use of the OpenIoT platform involves:

- The formulation of a request for an IoT service using the Request Definition tool, and its submission to the (Global) Scheduler component. The request specifies the needed sensors and the type of processing to be applied over the data, as well as the preferred visualization of the results.
- The parsing of the IoT service request by the scheduler, and the subsequent discovery of the sensors/ICOs to be used in order to deliver the IoT service. Toward discovering the required sensors, the Directory Service is queried and accessed.
- The formulation of the service (eg, in the form of a SPARQL query) and its persistence in the cloud, along with other metadata about the service. The metadata include a handle/identifier to the created IoT service.
- The execution of the service by end users (based on the handle of the target service) and the visualization of the results.

The platform caters to the optimization of the resources entailed in the delivery of IoT services. These optimizations leverage data formulated during the scheduling process, which is described in the following section, along with the functionalities of the Global Scheduler component.

2.4 SCHEDULING PROCESS AND IoT SERVICES LIFECYCLE

The Global Scheduler component is the main and first entry point for service requests submitted to the cloud platform. It parses each service request and accordingly performs two main functions toward the delivery of the service, namely the selection of the sensors/ICOs involved in the service, but also the reservation of the needed resources. The scheduler manages all the metadata of the IoT services, including: (1) The signature of the service (ie, its input and output parameters), (2) the sensors/ICOs used to deliver the service, and (3) execution parameters associated with the services, such as the intervals in which the service shall be repeated, the types of visualization (in the request presentation), and other resources used by the service. In principle, the Global Scheduler component keeps track of and controls the lifecycle of IoT services, which is depicted in Fig. 2.2. In particular, the following lifecycle management services are supported by the scheduler:



Register Command
Register
Unregistered
Available Sensors do not Fulfill The Criteria
Successfully Registered
Available Sensors do not Fulfill The Criteria
Update Command
Update Service
Registered
Unregister Command
Unregister (SPARQL cannot be executed)
Update Service Executed
Suspended
Service Successfully Updated
Disable Command
Available Sensors do not Fulfill The Criteria
Resources Updated
Service Successfully Enabled
Disable (SPARQL can be executed)
Update Service
Unregister Successfully Executed
Enable
Update (Resources) Executed
Update (Resources)
Enable Command
Repeat in a stable time interval defined by the system Admin
Available Sensors do not Fulfill The Criteria
Disabled

FIGURE 2.2 State Diagram of the OpenIoT Services Lifecycle Within the Scheduler Module

- *Resource Discovery*: This service discovers a virtual sensor's availability. It therefore provides the resources that match the requirements of a given request for an IoT service.
- *Register*: This service is responsible for establishing the requested service within the cloud database. To this end, it initially identifies and logs (within the cloud) all the sensors/ICOs, which are pertinent and/or needed for delivering the requested IoT service. The metadata of the IoT service (ie, signature, sensors/ICOs, and execution parameters) are persisted to the cloud, based on appropriate data structures. As part of the registration process, a unique identifier (ServiceID) is assigned to the requested IoT service. The current implementation of the "Register" service, as part of the OpenIoT Open Source Project, persists the service description as a SPARQL query (which covers a wide range of sensor queries/services against the W3C SSN directory service). Furthermore, the open source implementation maintains an appropriate set of data structures (within the cloud), which holds other metadata for each registered IoT service.
- *Unregister*: In the scope of the unregister functionality for given IoT service (identified through its ServiceID), the resources allocated for the service (eg, sensors used) are released (ie, disassociated from the service). In the case of an active service, a deactivation process is initially applied. The status of the service is appropriately updated in the data structures holding the metadata about the service.
- *Suspend*: As part of suspend functionality, the service is deactivated and therefore its operation is ceased. Note however, as part of the suspension the platform does not release the resources associated with the service.
- *Enable from Suspension*: This functionality enables a previously suspended service. The data structures holding the service's metadata in the cloud are appropriately updated.
- *Enable*: This service allows the enablement of an unregistered service. In practice, this functionality registers the service once again in the platform, through identifying and storing the required sensors/ICOs.
- *Update*: This service permits changes to the IoT service. In particular, it allows for the updating of the service's lifecycle metadata (ie, signature, sensors/ICOs, execution parameters) according to the requested changes. In the scope of the OpenIoT open source implementation, the scheduler formulates an updated service description (as SPARQL script), based on the updated user request. It also updates the data structures comprising the metadata of the service based on the updated information.
- *Registered Service Status*: This service provides the lifecycle status of a given IoT service (which is identified by its ServiceID). Detailed information (ie, all the metadata) about the IoT service is provided.
- *Service Update Resources*: This service checks periodically (at a configurable specified time-interval) all the enabled services, and identifies those using mobile sensors [eg, smartphones, UAVs (Unmanned Aerial Vehicles)]. Accordingly, it updates (if needed) the IoT service metadata on the basis of the newly defined sensors that support the IoT service. Such an update is needed in cases where a mobile sensor no longer fulfills the location-based criteria set by the service, or even in cases where additional (new) sensors fulfill these criteria.
- *Get Service*: This service retrieves the description of a registered service, that is, the SPARQL description in the case of the OpenIoT open source implementation.
- *Get Available Services*: This service returns a list of registered services that are associated with a particular user. Note that the various IoT services are registered and established by users of the platform.

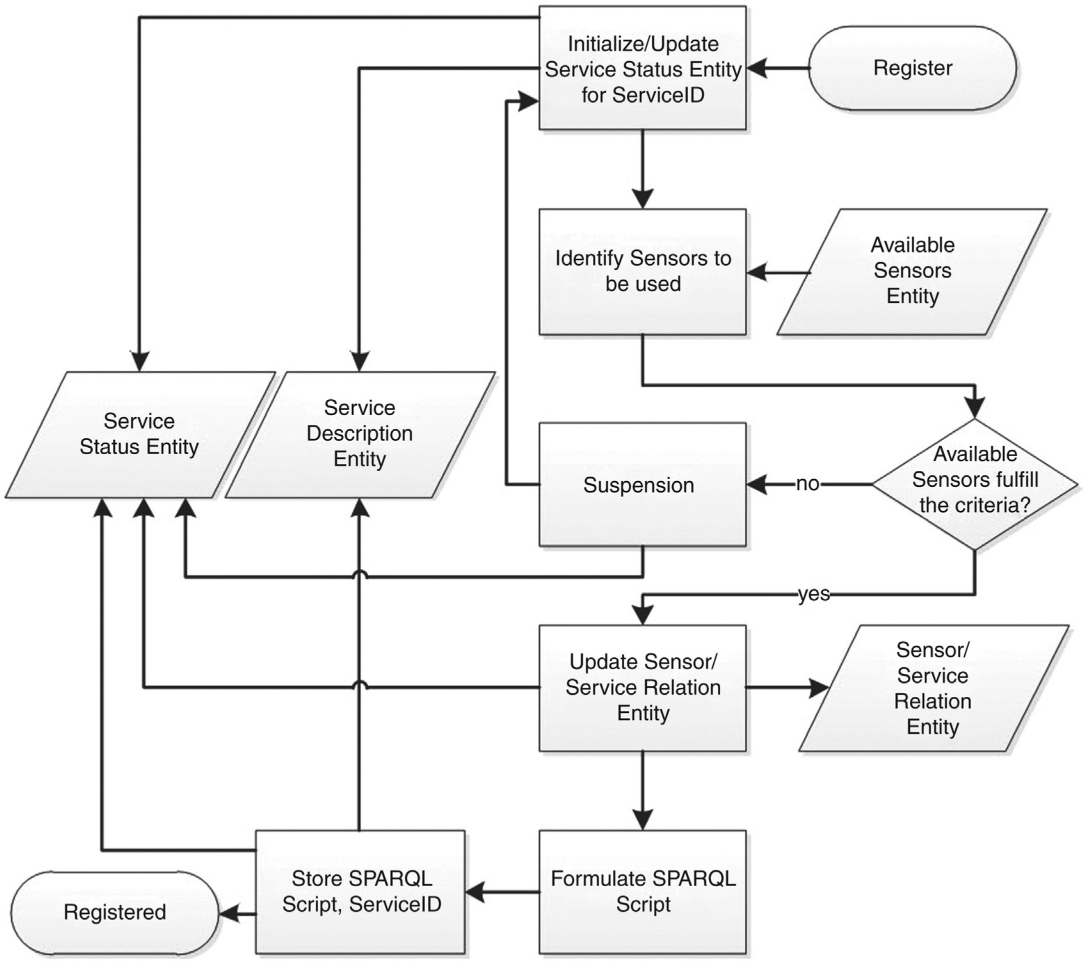

Initialize/Update Service Status Entity for ServiceID
Register
Identify Sensors to be used
Available Sensors Entity
Service Status Entity
Service Description Entity
Suspension
no
Available Sensors fulfill the criteria?
yes
Update Sensor/ Service Relation Entity
Sensor/ Service Relation Entity
Registered
Store SPARQL Script, ServiceID
Formulate SPARQL Script

FIGURE 2.3 "Register Service" Process Flowchart

In order to support the lifecycle services outlined previously, the scheduler provides an appropriate API, which also enables the services' transition between the various lifecycle states. The platform also supports baseline authentication and access-control mechanisms, which allows specific users to access resources and services available within the platform. Note that only registered users are able to leverage these aforementioned lifecycle management functionalities.

The following figures illustrate the details of some lifecycle-management use cases within the Scheduler component. In particular, Fig. 2.3 illustrates the main workflow associated with the service registration process. In the scope of this process, the Scheduler attempts to discover the resources (sensors, ICO) that will be used for the service delivery. In case there are no sensors/ICOs that can fulfill the request, the service is suspended. In case a set of proper sensors/IOCs is defined, the relevant data entities are updated (eg, relationship of sensors to services) and a SPARQL script associated with the service is formulated and stored for later use. Following the successful conclusion of this process, the servicer enters the "Registered" state and is available for invocation.

Likewise, Fig. 2.4 illustrates the process of updating the resources associated with a given service. As already outlined, such an update process is particularly important when it comes to dealing with IoT services that entail mobile sensors and ICOs, that is, sensors and ICOs whose location is likely to change within very short timescales (such as mobile phones and UAVs). In such cases, the Update Resources process could regularly check the availability of mobile sensors and their suitability for the registered service whose resources are updated. The workflow in Fig. 2.4 assumes that the list of mobile sensors is known to the service (ie, the sensors' semantic annotations indicate whether a sensor is mobile or not).

Even though the process/functionality of updating resources is associated with the need to identify the availability and suitability of mobile sensors, in principle the Update process can be used to update the whole list of resources that contribute to the given service. Such functionality helps the OpenIoT platform in dealing with the volatility of IoT environments, where sensors and ICOs may dynamically join or leave.

Finally, Fig. 2.5 illustrates the process of unregistering a service, in which case the resource associated with the service is released. The data structures of the OpenIoT service infrastructure are also modified to reflect the fact that the specified service is no longer using its resources. As already explained, this update is important for the later implementation of the resource management and optimization functionalities.

The previously outlined scheduling functionalities enable the delivery of the service through the Service Delivery and Utility Manager component of the OpenIoT architecture. The latter component has a dual functionality: On the one hand (as a service manager) it is the module enabling data retrieval from the selected sensors comprising the IoT service. On the other hand, the utility manager maintains and retrieves information structures regarding service usage, and supports metering, charging, and resource management processes.

The Service Delivery and Utility Manager (SD&UM) provides an appropriate API, enabling the platform to provide the outcome of a given service. In particular, the module supports:

- *Subscription for a report*: This service enables invocation of already defined IoT services (identified based on their ServiceID). In particular, it supports the collection of results from a given IoT service (eg, a SPARQL service) and their dispatching toward an application's destination address (URI).
- *Callback service*: This service is instantiated in order to deliver a report to all subscribers to a given IoT service (identified based on its ServiceID). The report is delivered according to the schedule defined by the user at the service registration time.
- *Unsubscribe for a report*: This service is invoked by the user in order to cease its subscription to a given report.
- *Poll for a report*: This service enables the user to periodically invoke an already defined IoT service within specified time intervals.
- *Get the utility usage of a user*: This service enables the user to retrieve utility information associated with a specific user. It takes into account all the IoT services that have been used by the specific user.
- *Get the utility usage of a registered service*: This service enables the user to retrieve the utility information associated with a specific IoT service (identified based on its ServiceID). Note that this functionality enables the application of utility functions over utility metrics, in order to calculate prices/costs in accordance to the accounting policies of the IoT/cloud service provider.

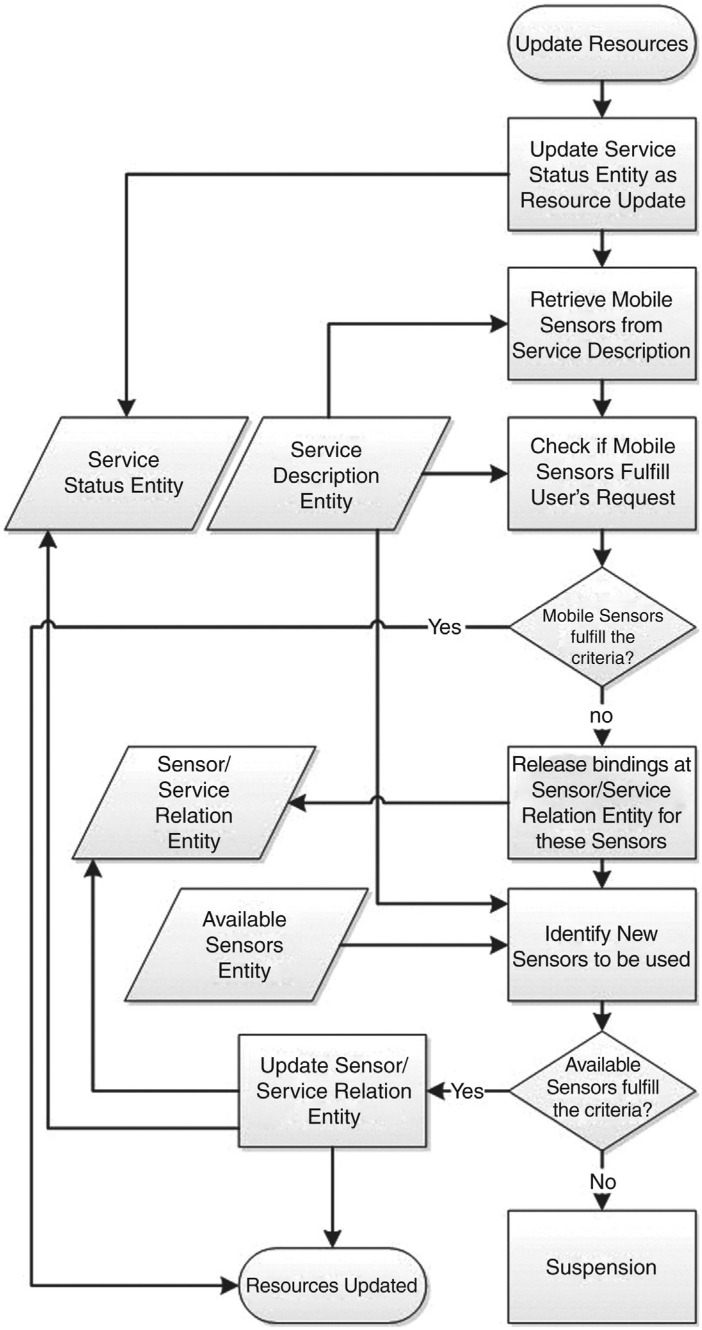

Update Resources
Update Service Status Entity as Resource Update
Retrieve Mobile Sensors from Service Description
Service Status Entity
Service Description Entity
Check if Mobile Sensors Fulfill User's Request
Mobile Sensors fulfill the criteria?
Yes
no
Sensor/ Service Relation Entity
Release bindings at Sensor/Service Relation Entity for these Sensors
Available Sensors Entity
Identify New Sensors to be used
Update Sensor/ Service Relation Entity
Yes
Available Sensors fulfill the criteria?
No
Resources Updated
Suspension

FIGURE 2.4 "Update Resources" Service Flowchart

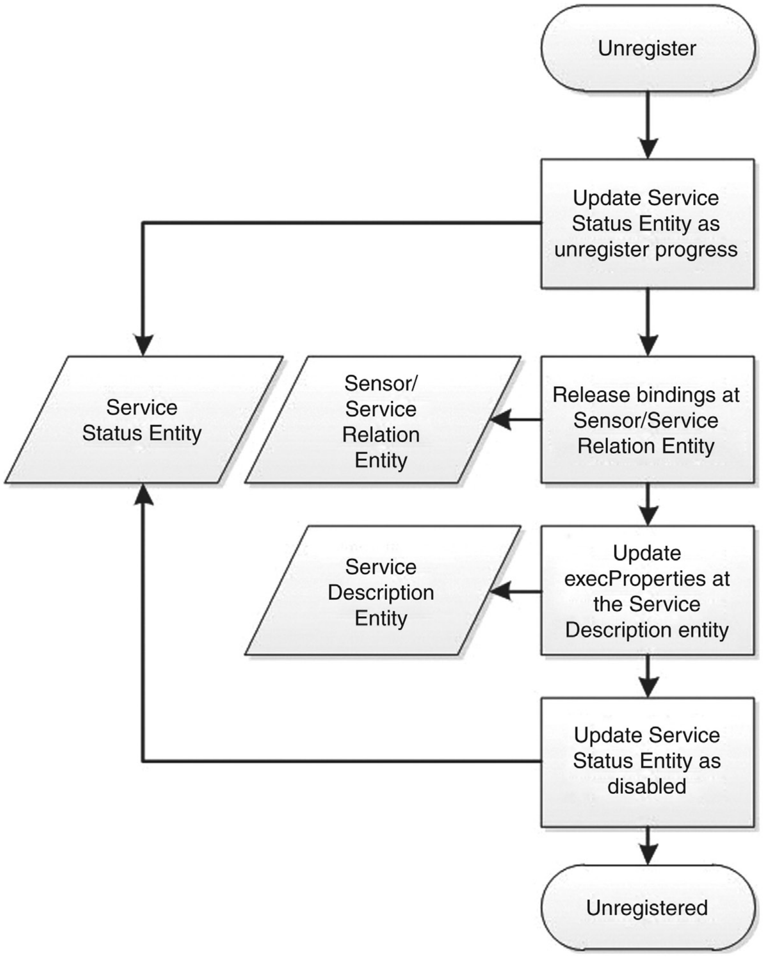

Unregister
Update Service Status Entity as unregister progress
Service Status Entity
Sensor/ Service Relation Entity
Release bindings at Sensor/Service Relation Entity
Service Description Entity
Update execProperties at the Service Description entity
Update Service Status Entity as disabled
Unregistered

FIGURE 2.5 "Unregister Service" Service Flowchart

- *Record utility usage*: This functionality enables the recording/logging of usage information, associated with an IoT service, in terms of volume of requested data and the types of sensors/resources used. Utility recording can be activated as a result of a polling request, as well as in the scope of a callback service.
- *Get service status*: This functionality provides access to the status of a specific IoT service by providing the ServiceID.
- *Get service description*: This functionality provides access to the description of the service.
- *Get available services*: This service functionality allows the user to access the list of IoT services, which are associated with a specific user.

As already outlined, the service delivery and utility management functionalities are based on the earlier scheduling of the requests for IoT services. In particular, the scheduling of requests ensures the logging of the appropriate information for managing service delivery to the various users, based on both push and pull functionalities (ie, supported by the “subscription” and “polling” models outlined previously). Furthermore, the management of scheduling information provides a basis for calculating utility metrics for metering, accounting, and billing purposes. With all of this information at hand, the introduced, converged IoT/cloud infrastructure provides the means for implementing resource management and optimization algorithms, which are outlined in the following section.

2.5 SCHEDULING AND RESOURCE MANAGEMENT

The OpenIoT scheduler enables the availability and provision of accurate information about the data requested by each service, as well as about the sensors and ICOs that will be used in order to deliver these data. Hence, a wide range of different resource management and optimization algorithms can be implemented at the scheduler component of the OpenIoT architecture. Furthermore, the envisaged scheduling at the local (ie, sensor middleware) level enables resource optimization at the sensor data-acquisition level (ie, at the edges of the OpenIoT infrastructure).

In terms of specific optimizations that can be implemented over the introduced IoT/cloud infrastructure are optimization techniques that have their roots in the WSN literature, where data management is commonly applied as a means to optimize the energy efficiency of the network [15]. In particular, the scheduler components (at both the global and the local levels) maintain information that could optimize the use of the network on the basis of aggregate operations [16], such as the aggregation of queries to the sensor cloud [17]. Furthermore, a variety of in-network processing and data management techniques can be implemented in order to optimize processing times and/or reduce the required access to the sensor network [18,19]. The criteria for aggregating queries and their results could be based on common spatial regions (ie, data aggregation based on sensors residing in geographical regions of high interest). In general, the in-network processing approaches previously outlined can be classified into three broad categories, namely push [20], pull [16,19], and hybrid approaches [18,21].

Another class of optimizations that are empowered by the introduced scheduling approach are caching techniques [22,23], which typically reduce network traffic, enhance the availability of data to the users (sink), and reduce the cost of expensive cloud-access operations (eg, I/O operations to public clouds). The caching concept involves maintaining sensor (data streams) data to a cache memory in order to facilitate fast and easy access to them. In the case of OpenIoT, caching techniques could obviate the need to execute the results of previously executed SPARQL queries.

As a proof-of-concept implementation, we have implemented two resource optimization schemes over the open source OpenIoT cloud platform. The first scheme concerns bandwidth and storage optimization through indirect control over the sensor, and falls in the broader class of in-network process approaches discussed previously. As part of this scheme, a pull approach is adopted in terms of accessing the sensor networks of the various nodes. In particular, a periodic timed (ie, polling) task is running on the X-GSN module, which is responsible for direct sensor management. This task queries the LSM/W3C SSN repository, in order to determine which sensors are needed and used by IoT services. The task compares the query results to the LSM (triplets), with the list of sensors that are currently active on the X-GSN sensor middleware module. Accordingly, X-GSN: (1) activates sensors that have needed/used an IoT service, which are not active on the module; and

(2) deactivates the sensors that are active on the module, but have not been used by any IoT service. This process is illustrated on the sequence diagram in Fig. 2.6. The implementation of this scheme ensures that no unnecessary data will be streamed from the sensors (and the X-GSN node that they are attached to) to the cloud, thereby saving in terms of bandwidth and costs associated with cloud access. Hence, the implemented pull approach can serve as a basis for optimizing both latency and monetary costs.

The resource optimization scheme was implemented on the basis of the caching of SPARQL queries/requests. Caching can alleviate one of the most significant drawbacks of the use of triple stores for the deployment of semantic technologies, which is their low performance compared to conventional relational databases. Besides the issue of performance, caching can also have monetary benefits, given that accessing remote data-stores like Amazon S3 or Google Cloud Data-Store incurs pay-as-you-go costs, depending on the provider's pricing scheme.

The caching solution that was implemented over the OpenIoT infrastructure is based on [24]. In particular, a small proxy layer is implemented and used to route all SPARQL queries. Whenever a query is entered into the system, the proxy layer checks whether the result has already been cached. In such a case, the result is returned to the client directly through the cache without accessing the SPARQL data store. In any other case the query is redirected to the SPARQL data store and the result is stored in the local cache before it is returned to the user (client).

As a validating example of the proxy layer implementation over the OpenIoT sensor cloud, we have a assumed a Pareto distribution for the probability density of the various SPARQL queries, as in [24], which is more practical compared to the assumption that all queries arrive with the same probability (as in the [25] benchmark). Note that the "*a*" parameter of the Pareto distribution $p(x) = \frac{ab^a}{x^{a=1}}$ allows the simulation of either a wider or narrower spectrum of repeated queries. As expected, the wider the variety of SPARQL queries (representing IoT services), the fewer unique queries that are serviced directly from the LSM implementation in the cloud, resulting ultimately in the majority of queries being serviced from the cache proxy. In order to simulate a practical scenario, we assume that the Amazon S3 public cloud data-store [which features a linear pricing-scheme (eg, \$0.005/1000 requests)] is used in conjunction with the OpenIoT infrastructure. We also assume that the cache miss-rates are those depicted in [24] for a benchmark based on 10 million triples and 12,500 queries. Furthermore, yearly server operational costs that support a 20TB cache have been taken into account (Table 2.1). This cache capacity is considered sufficient to store all the query results obtained from the cloud data-store for this particular scenario.

Under these specific circumstances, Fig. 2.7 illustrates the total costs incurred for seven different scenarios of the SPARQL queries distribution that correspond to different varieties of queries. The different varieties correspond to different parameters of the Pareto distribution in [24], which result in different miss-rates [25].

As expected, for a low number of requests per hour, there is no benefit for using cache. At a medium-high number of hourly requests, such as the second category at 1450 Krph, the threshold where it becomes more efficient to use a caching solution is just hit. It is finally evident, in the last category at 2000 Krph, that at a high number of requests it is far more efficient to use cache. In order to achieve an efficient caching solution there must be a clear estimate, first of all, of the average requests per hour on the cloud data-store, as well as to what extent the cache storage capacity is sufficient. Finally, it is also evident by this simulation that the determining factor for cache performance

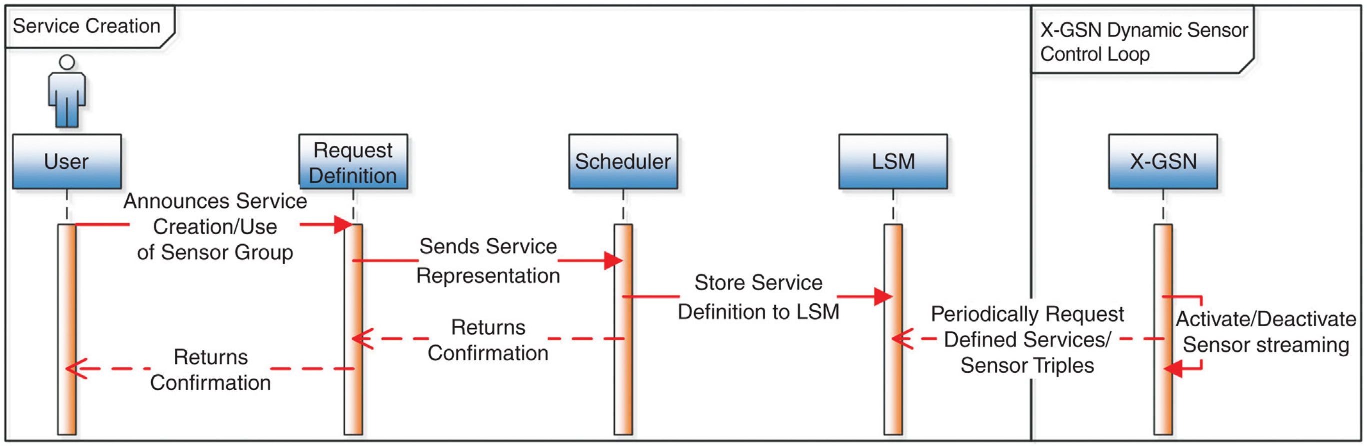

Service Creation
X-GSN Dynamic Sensor Control Loop
User
Request Definition
Scheduler
LSM
X-GSN
Announces Service Creation/Use of Sensor Group
Sends Service Representation
Store Service Definition to LSM
Returns Confirmation
Returns Confirmation
Periodically Request Defined Services/ Sensor Triples
Activate/Deactivate Sensor streaming

FIGURE 2.6 **“Service Creation” Service Flowchart**

Table 2.1 Total Cost of Ownership for a 20TB Cache Server

Caching Server Cost/Unit	
Server Disk Capacity (TB)	5
Unit Cost (€)	3,500
Lifespan (years)	3
PV Discount Rate (%)	5
Server Maintenance/Year (€)	1,500
Energy cost/year (€)	1,000
Server Cost/Year (Present Value)	
Cache Size Required (TB)	20
Servers Required	4
Cost/Year (PV)	22,635.00 €

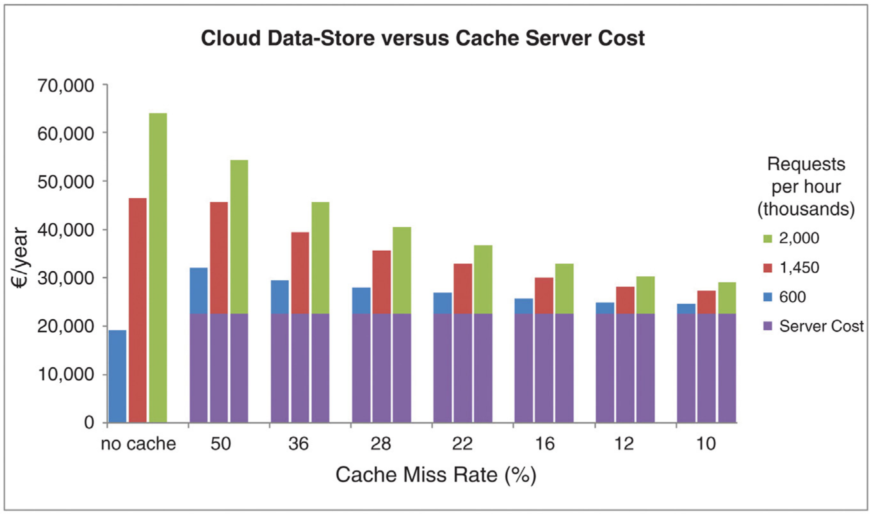


FIGURE 2.7 Comparison of Costs Associated With the Use of Cache Server to the Cost of the Use of a Public Cloud Data-Store

is not the absolute number of queries. Rather, it is the variety of different queries that are performed on the cloud data-store in order to quickly build up the cache. Hence, efficient implementations of the caching schemes could consider caching criteria that could minimize the spectrum of queries variance, thereby maximizing the hit ratios [eg, criteria based on temporal parameters (such as the time that the queries are asked) or spatial parameters (such as the location of the sensors entailed in the queries)].

2.6 VALIDATING APPLICATIONS AND USE CASES

The overall scheduling infrastructure has also been validated through the development of proof-of-concept IoT applications, which comprise multiple IoT services that have been integrated based on the OpenIoT infrastructure. One of the proof-of-concept applications falls in the wider realm of smart-city applications, aimed at providing a set of smart services within a student campus in Karlsruhe, Germany. In particular, this smart-campus application creates a semantic storage with an ontology-based description of real-world objects, which are identified by different labeling methods (QR-Codes, RFID-Tags). The objects are associated to their semantic description by unified resource identifiers and stored to the semantic storage. The semantic information about real-world objects is retrieved and updated via semantic queries (SPARQL), which are invoked through mobile devices. The information about the real-world objects (eg, rooms, books) is then displayed by dedicated views, and within existing views of standard smartphone applications, like a map application.

The second use-case concerns the implementation of IoT services for manufacturing, and, more specifically, for the printing and packing industry, with emphasis on the production of boxes. The IoT services support key production processes in this industry, such as printing on paper sheets and die-cutting (for perforation of the sheets), as well as gluing and folding the pieces of a box. A variety of sensors are employed to facilitate production-line automation and quality-control checks, including laser sensors, high-speed 1D/2D barcode verification cameras, weight sensors, contrast and color sensors (for marking code identification), as well as ultrasonic sensors (for measuring heights and material-reel diameters). In this environment, the OpenIoT infrastructure is used to enable the dynamic on-demand formulation, calculation, and visualization of KPIs (Key Performance Indicators) about the manufacturing processes. Interesting KPIs include, for example: (1) in the area of materials consumption, the rate of consumption and how much scrap is produced; (2) in the area of machine performance, how fast each machine is working, what is the rate of product/shipping container production, and the overall efficiency of the machines; (3) in the area of labor activity and performance, how much time is spent setting up/repairing the machine; and (4) in the area of machine operation, an interesting KPI relates to tracking the time that machines spend in their various modes (ie, setup/repair/idle/operation). To this end, high-level events captured, based on the processing of the aforementioned sensors, are announced as virtual sensors to the W3C SSN directory through the X-GSN middleware. In particular, KPI calculations are implemented as a set of X-GSN virtual sensors, and, accordingly, are published to the sensor cloud and made available for semantic querying. Requests to the OpenIoT system (via the scheduler) are able to define, select, filter, and visualize KPIs on the basis of various selection criteria, including location (plant, floor, type). At the same time, several other requests will be able to compose (eg, aggregate) KPIs on the basis of other, more elementary, KPIs. Note that composite KPIs could dynamically combine information from multiple machines and plants, as soon as these are published to the sensor cloud. Some examples of composite KPIs formulated and measured include: (1) find the rates of all machines in a company's factories (within location L1, L2, L3) and plot them together; (2) find the operation status of all machines of type X in a company's factories (within locations L1, L2, L3) and overlay them; and (3) find the rates of all machines in my factories (within locations L1, L2, L3) and the local temperatures at locations (L1, L2, L3), and plot them (eg, in order to understand how local temperature variations affect machine operation).

The third use-case (part of the "Phenonet" project) uses state-of-the-art sensor network technology, in order to gather environmental data for crop-variety trials, at a far higher resolution than conventional methods, and provides a high-performance real-time online data-analysis platform that allows scientists and farmers to visualize, process, and extract both real-time and long-term crop performance information. Phenonet uses a WSN in a wheat-variety trial at the Yanco Managed Environment Facility in New South Wales (NSW). This was a key part of an experiment to test the feasibility of using remote monitoring of crop performance in trials across Australia. The WSN consists of sensors measuring (1) local environmental data, including: solar radiation, air temperature, relative humidity, rainfall, and wind speed; and (2) crop performance data, including: soil moisture, soil temperature, and an infrared sensor that measures leaf (crop canopy) temperature. The sensors are linked by short-range digital radio to a base station that can return the results in real time to a server in Canberra via 3G wireless networking. The raw data are disseminated to the sensor cloud, and are made available for normalization, integration correlation, and real-time visualization.

2.7 FUTURE RESEARCH DIRECTIONS

An open source implementation of the introduced scheduling concepts is available as part of the OpenIoT Open Source Project. We expect this chapter to motivate the open-source community toward providing other implementations of scheduling concepts for IoT services, including their integration with cloud computing infrastructures. Such concepts could deal with caching of IoT service requests and response data, taking into account the spatiotemporal characteristics of the respective IoT queries. In terms of the OpenIoT middleware infrastructure, future research work includes the development of tools for visualizing IoT resources, as part of the Integrated Development Environment (IDE) of the OpenIoT project.

2.8 CONCLUSIONS

In this chapter we have introduced the OpenIoT architecture as a practical approach to the integration and convergence of IoT with cloud computing. As part of this architecture we have also illustrated the benefits of a scheduling approach for IoT services. Scheduling is one of the key merits and distinguishing characteristics of OpenIoT, when compared to state-of-the-art approaches for IoT/cloud integration. Indeed, the presented approach allows for the logging of a wide range of information associated with IoT services, which can enable the implementation of utility-based functionalities, as well as a wide range of resource optimizations, both of which are essential to both end-users and cloud-service providers. In most cases these optimizations are also very important in order to fully leverage the benefits of a cloud computing model for IoT services, such as cost efficiency and the adoption of utility-driven pay-as-you-go operational models.

The merits of the introduced scheduling architecture have been validated based on practical implementations of use cases, as well as of resource-optimization mechanisms. In terms of use cases, three distinct IoT applications have been developed, which utilize a variety of sensors and IoT services across three different application domains (smart cities, manufacturing, crop management). At the same time, the resource optimization capabilities have been validated on the basis of the implementation of two different schemes, one relating to in-network optimization of sensor information dissemination and another to caching of semantic-based requests (ie, queries) for IoT services.

ACKNOWLEDGMENTS

Part of this work has been carried out in the scope of the OpenIoT project (FP7-287305) (http://openiot.eu). The authors acknowledge help and contributions from all partners of the project. The smart campus, manufacturing, and crop-management use cases that have been briefly presented in Section 2.5 have been implemented by the OpenIoT consortium partners Fraunhofer-IOSB (http://www.iosb.fraunhofer.de/), SENSAP S.A (www.sensap.eu), and CSIRO (csiro.au), respectively.

REFERENCES

[1] McFedries P. The cloud is the computer. IEEE Spectr 2008;45(8):p. 20.
[2] Vermesan O, Friess P. Internet of Things—global, technological, and societal trends. River Pub Ser Commun 2011.
[3] Tham CK, Buyya R. SensorGrid: integrating sensor networks and grid computing. CSI Commun 2005;29:24–9.
[4] Gaynor M, Moulton SL, Welsh M, LaCombe E, Rowan A, Wynne J. Integrating wireless sensor networks with the grid. IEEE Internet Comput 2004;8:32–9.
[5] Lim HB, et al. Sensor grid, integration of wireless sensor networks and the grid. In: IEEE conference on local computer networks, 30th anniversary (LCN'05); 2005.
[6] Lim HB, Ling KV, Wang W, Yao Y, Iqbal M, Li B, et al. The national weather sensor grid. In: Proceedings of the fifth ACM conference on embedded networked sensor systems (SenSys 2007); 2007.
[7] Lee K. Extending sensor networks into the cloud using Amazon web services. In: IEEE 2010 international conference on networked embedded systems for enterprise applications; 2010.
[8] Mehedi Hassan M, Song B, Huh E-n. A framework of sensor-cloud integration opportunities and challenges. ICUIMC 2009;618–626.
[9] Fox GC, Kamburugamuve S, Hartman R. Architecture and measured characteristics of a cloud based Internet of Things, API workshop 13-IoT Internet of Things, machine to machine and smart services applications (IoT 2012). In: The 2012 international conference on collaboration technologies and systems (CTS 2012); 2012. p. 21–25.
[10] Soldatos J, Serrano M, Hauswirth M. Convergence of utility computing with the Internet of Things. In: International workshop on extending seamlessly to the Internet of Things (esIoT) 2012 IMIS international conference. Palermo, Italy, July 4–6, 2012.
[11] Kranz M, Roalter L, Michahelles F. Things that twitter: social networks and the Internet of Things, what can the Internet of Things do for the citizen (CIoT). In: Proceedings of the workshop at the eighth international conference on pervasive computing. Helsinki, Finland, 2010.
[12] Aberer K, Hauswirth M, Salehi A. Infrastructure for data processing in large-scale interconnected sensor networks. MDM 2007;198–205.
[13] Chatzigiannakis I, Mylonas G, Sotiris E. Nikoletseas, 50 ways to build your application: a survey of middleware and systems for wireless sensor networks. ETFA 2007;466–73.
[14]Taylor K. Semantic sensor networks: the W3C SSN-XG ontology and how to semantically enable real time sensor feeds. In: Proceedings of the semantic technology conference. San Francisco, CA, USA, June 5–9, 2011.
[15] Abadi DJ, Madden S, Linder W. REED: robust, efficient filtering and event detection in sensor networks. In: Proceedings of the thirty-first VLDB conference. Trondheim, Norway; 2005. p. 769–780.
[16] Yao Y, Gehrke J. The cougar approach to in-network query processing in sensor networks. SIGMOD Rec 2002;31(3):9–18.
[17] Meng M, Yang J, Xu H, Jeong B-S, Lee Y-K, Lee S. Query aggregation in wireless sensor networks. IJMUE 2008;3(1)19–26.
[18] Lee KCK, Lee WC, Zheng B, Winter J. Processing multiple aggregation queries in geo-sensor networks. In: Proceedings of the eleventh international conference on database systems for advanced applications (DASFAA); 2006. p. 20–34.

[19] Madden SR, Franklin MJ, Hellerstein JM, Hong W. Tinydb: an acquisitional query processing system for sensor networks. ACM Trans Data Base Syst (TODS) 2005;30(1):122–73.
[20] Ye F, Luo H, Cheng J, Lu S, Zhang L. In: A two-tier data dissemination model for large-scale wireless sensor network. In: MobiCom 2002: Proceedings of the eighth annual international conference on mobile computing and networking. Atlanta, GA: ACM; 2002. p. 148–159.
[21] Li X, Huang Q, Zhang Y., Combs, needles, haystacks: balancing push and pull for discovery in large-scale sensor networks, ACM sensys. Baltimore, MD; 2005.
[22] Li S, Zhu LJ. Data caching based queries in multi sink sensor networks. In: IEEE proceedings of the fifth international conference on mobile ad-hoc and sensor networks; 2009.
[23] Chow CY, Leong HV, Chan ATS. GroCoca: group-based peer-to-peer cooperative caching in mobile environment. IEEE J Sel Areas Commun 2007;25(1)179–191.
[24] Martin M, Unbehauen J, Auer S. Improving the performance of semantic web applications with SPARQL query caching. In: Proceedings of the extended semantic web conference (ESWC). Heraklion, Greece: Springer; 2010. p. 304–318.
[25] Bizer C, Schultz A. The berlin SPARQL benchmark. Int J Sem Web Inf Syst 2009;5(2):1–24.