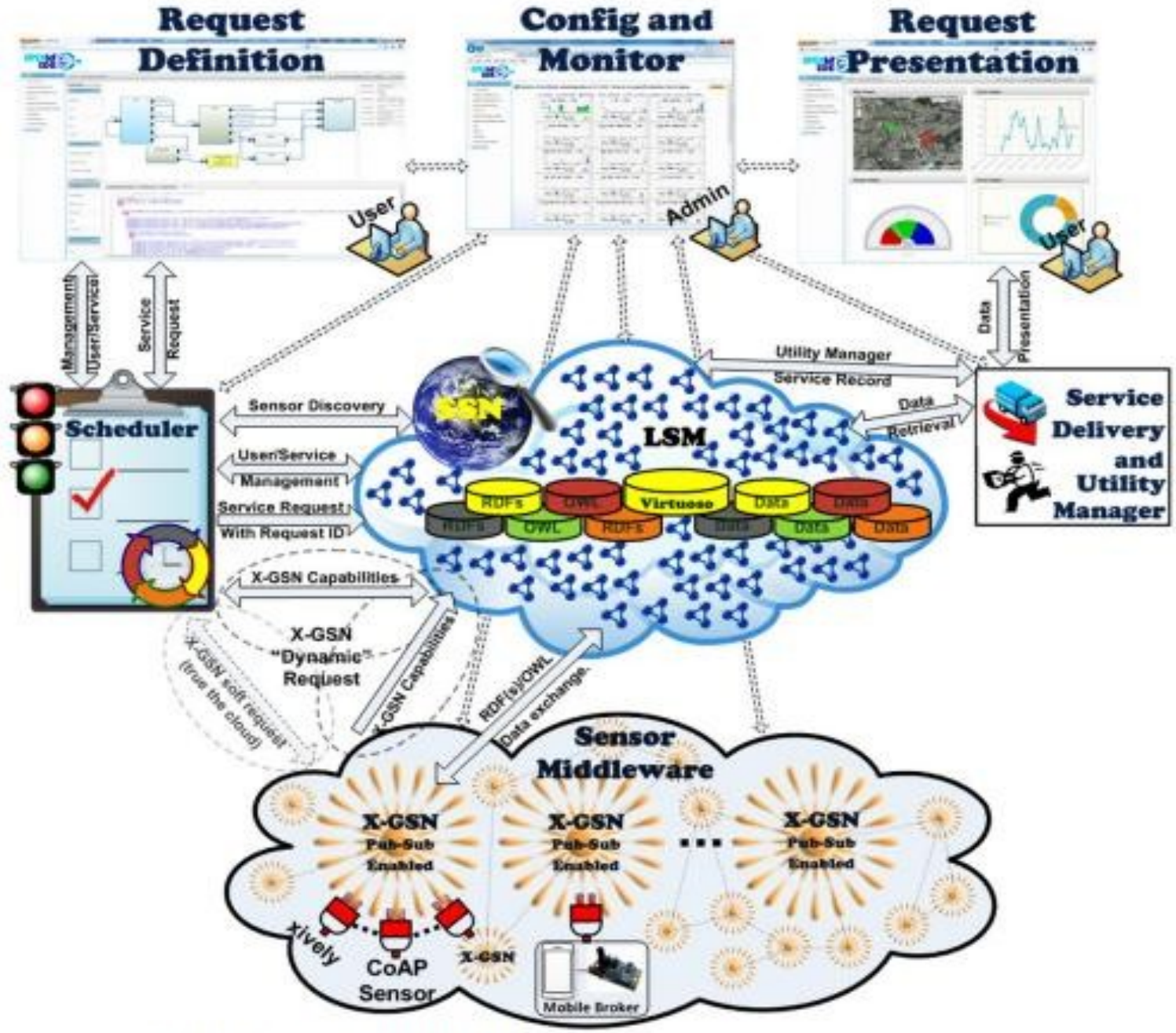


OpenIoT Architecture for IoT/Cloud Convergence

OPEN IoT ARCHITECTURE FOR IoT/CLOUD CONVERGENCE(Pg.34)

- The various 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.
- The IoT paradigm relies on the identification and use of a large number of heterogeneous physical and virtual objects
- Multisensor applications need to perform complex processing that is subject to timing and other QoS constraints
- The implementations of both technologies, has become apparent that their convergence could lead to a range of multiplicative benefits.

OPEN IoT ARCHITECTURE FOR IoT/CLOUD CONVERGENCE(Pg.34)

- Our approach for converging IoT and cloud computing is reflected in the OpenIoT architecture fig above
- The Sensor Middleware, which collects, filters, and combines data streams stemming from virtual sensors 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, it enables the access to information stemming from the real world.

OPEN IoT ARCHITECTURE FOR IoT/CLOUD CONVERGENCE(Pg.34)

- 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

OPEN IoT ARCHITECTURE FOR IoT/CLOUD CONVERGENCE(Pg.34)

- The Directory Service, which stores information about all the sensors that are available in the OpenIoT platform.
- It also provides the 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.
- Semantic Web techniques (eg, SPARQL and RDF) and ontology management systems (eg, Virtuoso) are used for querying the directory service.

OPEN IoT ARCHITECTURE FOR IoT/CLOUD CONVERGENCE(Pg.34)

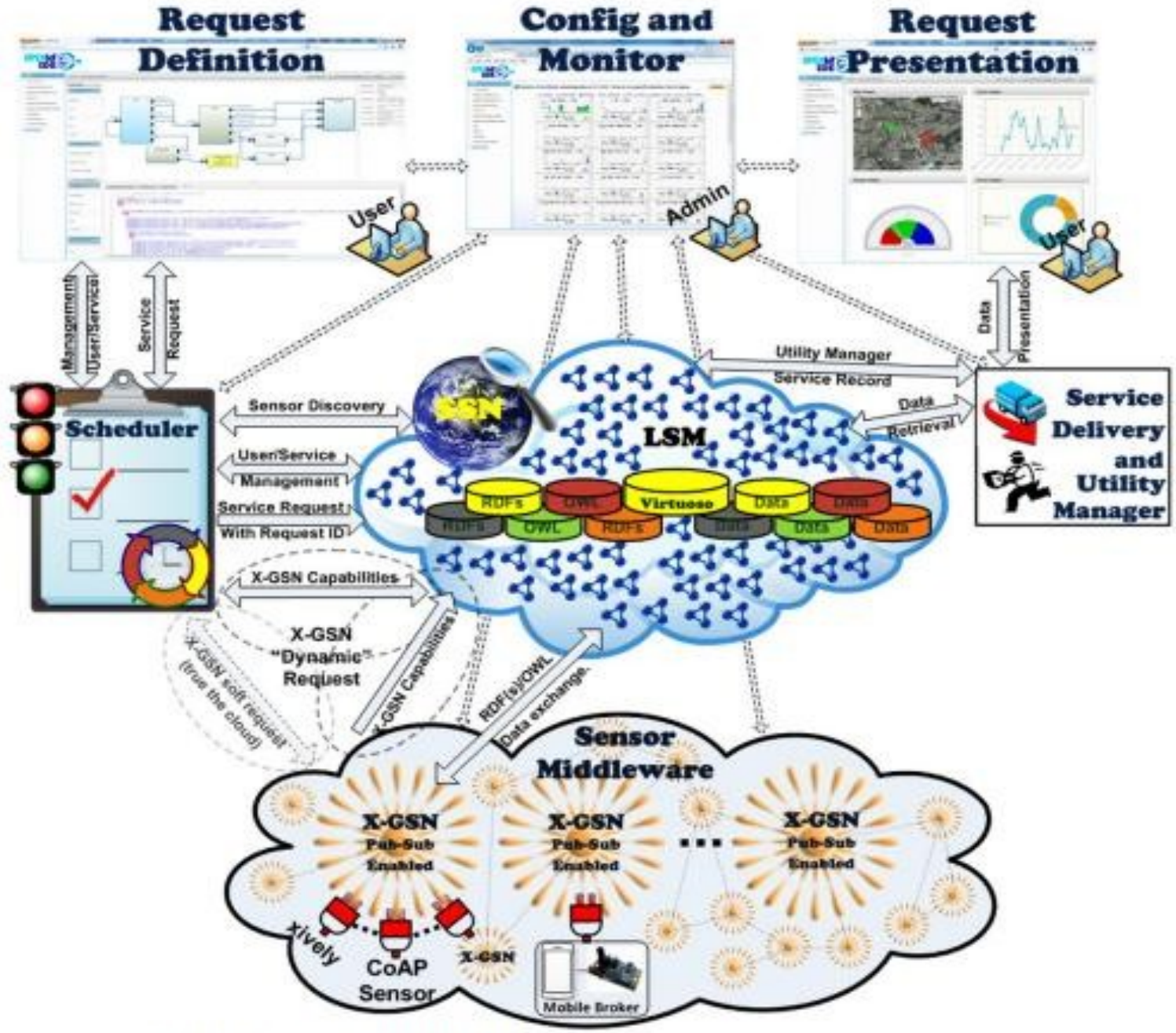
- The Service Delivery and Utility Manager, which performs a dual role. On the one hand, it combines the data streams in order to deliver the requested service.
- On the other hand, this component acts as a service-metering facility, which keeps track of utility metrics for each individual service.
- 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.

OPEN IoT ARCHITECTURE FOR IoT/CLOUD CONVERGENCE(Pg.34)

- Global Scheduler enables the scheduling of all IoT services
- This component undertakes the task of parsing the service request, and, accordingly, discovering the sensors that can contribute to its fulfillment
- This component ensures the proper access to the resources (eg, data streams) that a service require.
- The Local Scheduler component, which is executed at the level of the Sensor Middleware. This ensures the optimized access to the resources managed by sensor middleware instances.
- The Global Scheduler regulates the access to the resources of the OpenIoT platform (notably the data streams residing in the cloud), where as ,its local counterpart regulates the access and use of the data streams at the lower level of the Sensor Middleware.

OPEN IoT ARCHITECTURE FOR IoT/CLOUD CONVERGENCE(Pg.34)

- The Request Definition tool It enables the specification of service requests to the Open IoT platform. Ie. formulating requests to global scheduler
- 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.
- The Configuration and Monitoring component, which enables management and configuration functionalities over the sensors, and the IoT services that are deployed within the platform.



OpenIoT Architecture for IoT/Cloud Convergence

Workflow of an Open IoT Platform

- **The formulation of a request for an IoT service** using the Request Definition tool, and its submission to the (Global) Scheduler component.
- **The parsing of the IoT service request** by the scheduler, and the subsequent discovery of the sensors 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 execution of the service by end users and** the visualization of the results.

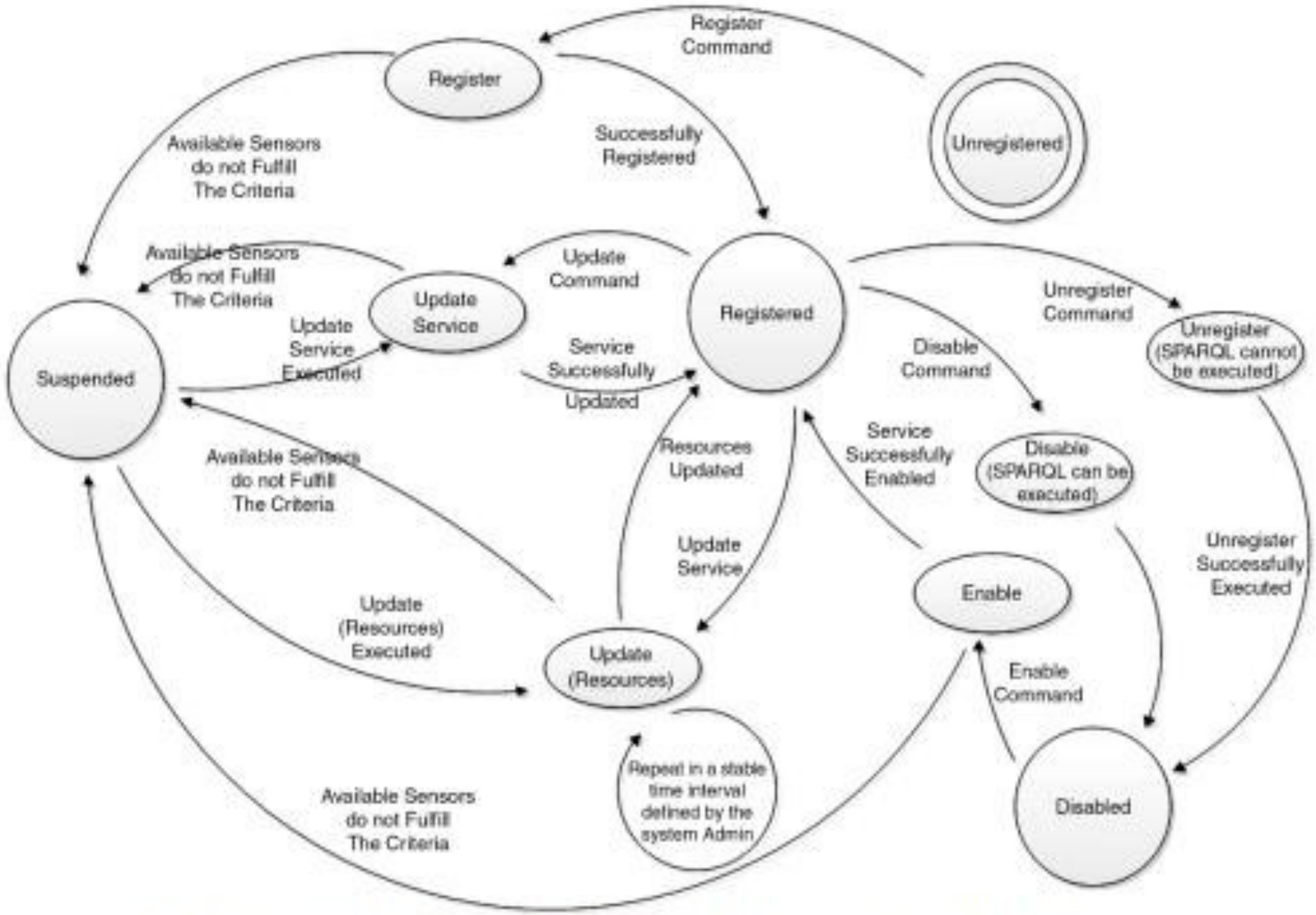
Scheduling Process and IoT Services Lifecycle

- **The Global Scheduler component** is the main 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 (selection of the sensors, the reservation of the needed resources)
- The scheduler manages all the metadata of the IoT services:

 The signature of the service (ie, its input and output parameters),

 The sensors used to deliver the service

 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.



State Diagram of the OpenIoT Services Lifecycle Within the Scheduler Module

Lifecycle management services are supported by the scheduler:

- ***Resource Discovery***: This service discovers a 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-a unique identifier (ServiceID) is assigned to the requested IoT service.
  This implementation maintains an appropriate set of data structures 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 are released** -The status of the service is appropriately updated in the data structures holding the metadata about the service.

Lifecycle management services are supported by the scheduler:

- ***Suspend***: The service is deactivated and therefore its operation is ceased- it 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- It registers the service once again in the platform, through identifying and storing the required sensors.

Lifecycle management services are supported by the scheduler:

- ***Update*:** This service permits changes to the IoT service- it allows for the updating of the service's lifecycle metadata (ie, signature, sensors, execution parameters) according to the requested changes.
  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.

Lifecycle management services are supported by the scheduler:

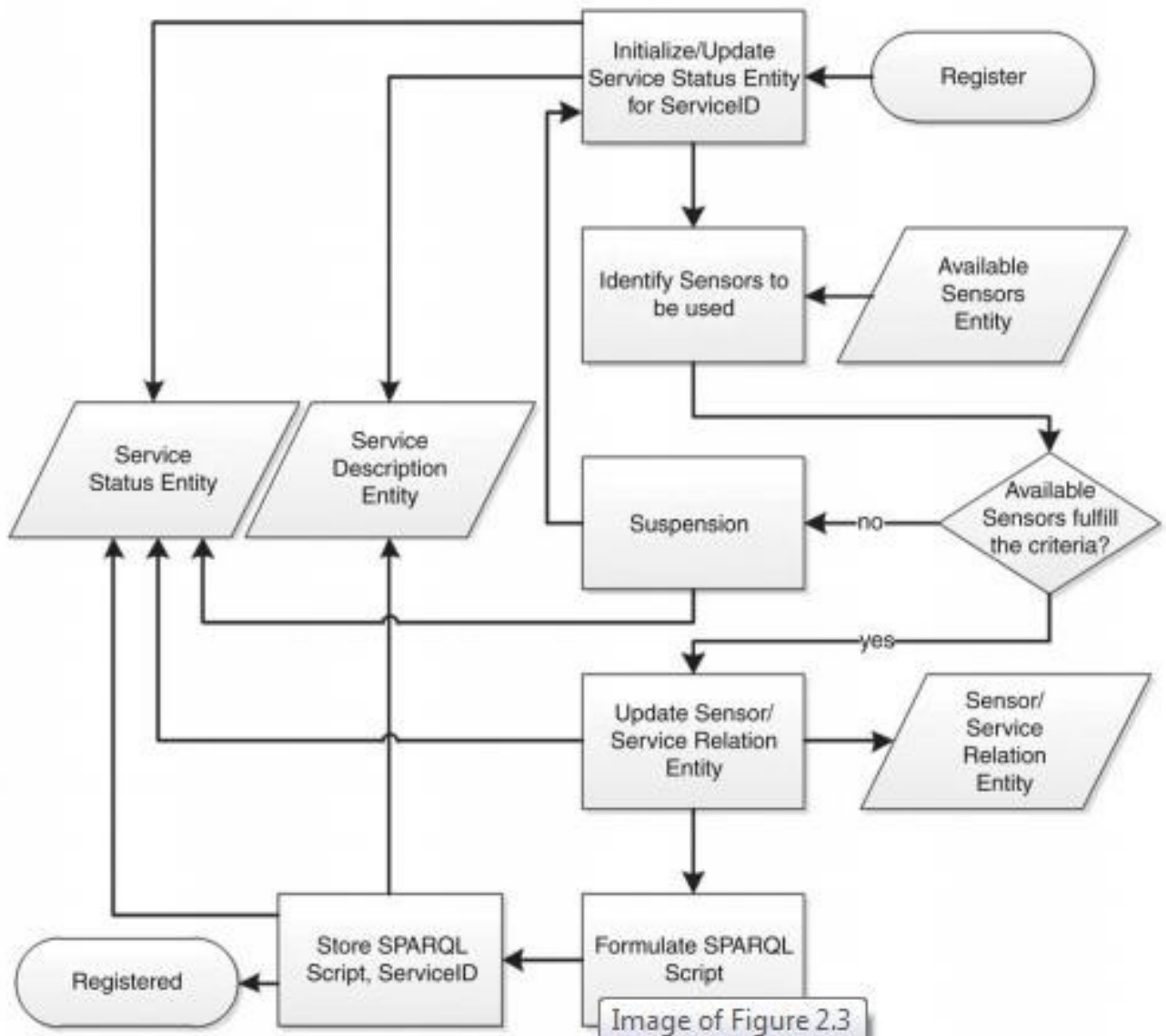
- ***Service Update Resources*:** This service checks periodically all the enabled services, and identifies those using mobile sensors

  eg, smart phones ,UAVs (Unmanned Aerial Vehicles).

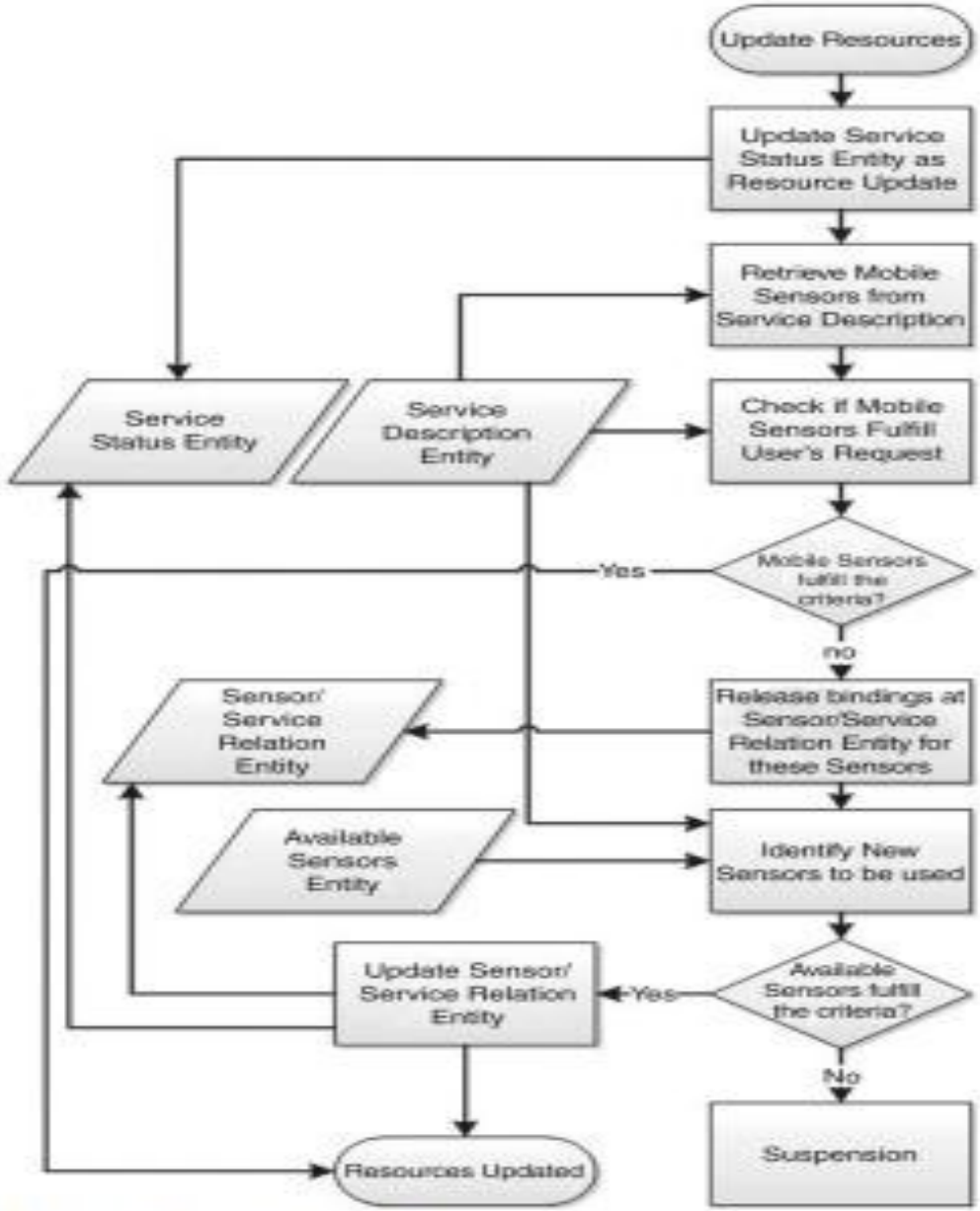
  Accordingly, it updates the IoT service metadata on the basis of the newly defined sensors that support the IoT service.

- ***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.

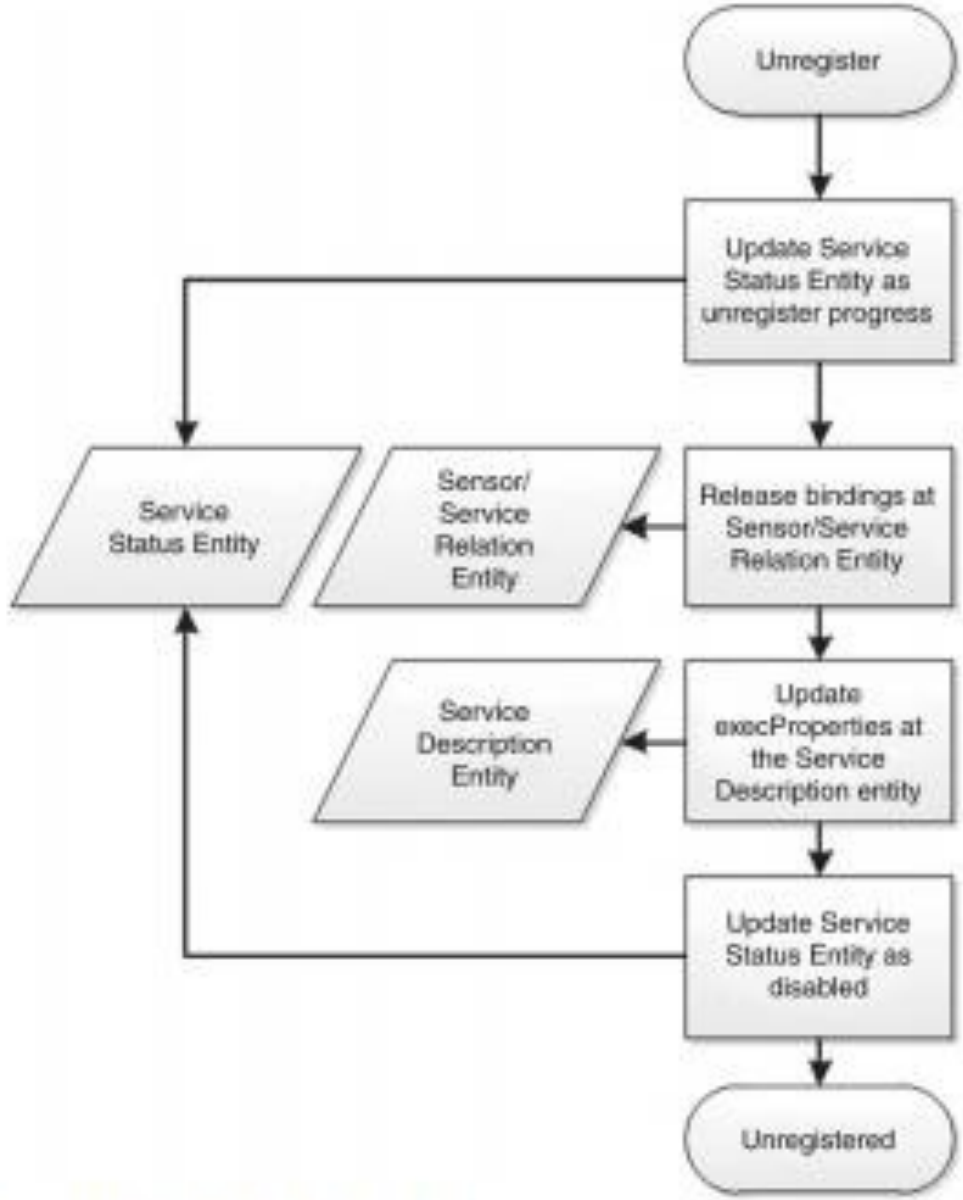
- ***Get Available Services*:** This service returns a list of registered services that are associated with a particular user(the various IoT services are registered and established by users of the platform)



"Register Service" Process Flowchart



"Update Resources" Service Flowchart



Unregister Service" Service Flowchart

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 that will be used in order to deliver these data.

- A wide range of different resource management and optimization algorithms can be implemented at the scheduler component of the OpenIoT architecture.

- Scheduling at the local level (sensor middleware) enables resource optimization at the sensor data acquisition level (at the edges of the OpenIoT infrastructure)

Resource optimization scheme

- A variety of **in-network processing and data management techniques**(push, pull and hybrid approaches) can be implemented in order to optimize processing times and/or reduce the required access to the sensor network.
- 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)
- Another class of optimizations that are empowered by the scheduling approach are **caching techniques**

…Resource optimization scheme

- The in-network process approaches optimization scheme concerns bandwidth and storage optimization through indirect control over the sensor. As part of this scheme, a pull approach is adopted in terms of accessing the sensor networks of the various nodes.

- A periodic 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, with the list of sensors that are currently active on the X-GSN sensor middleware module.

…Resource optimization scheme

- X-GSN:

(1) activates sensors that have needed/used an IoT service, which are not active on the module

(2) deactivates the sensors that are active on the module, but have not been used by any IoT service.

- The implementation of this scheme ensures that no unnecessary data will be streamed from the sensors to the cloud, thereby saving in terms of bandwidth and costs associated with cloud access.
- This pull approach can serve as a basis for optimizing both latency and monetary costs.

Caching Techniques

The caching concept involves **maintaining sensor (data streams) data to a cache memory** in order to facilitate fast and easy access to them.

- It reduce network traffic,
- enhance the availability of data to the users
- reduce the cost of expensive cloud-access operations (eg: I/O operations to public clouds).

…Caching Techniques

- The caching solution that was implemented over the OpenIoT – **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).

…Caching Techniques

- In order to achieve an efficient caching solution there must be a clear estimate of the **average requests per hour on the cloud data-store**, as well as to what extent the **cache storage capacity** is sufficient.

- 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