20MCA281- INTERNET OF THINGS

MODULE 1 – Part 2

Prepared By: Gloriya Mathew, Asst. Professor, Amal Jyothi College Of Engineering.

Internet of Things

Principles and Paradigms

Edited by **Rajkumar Buyya & Amir Vahid Dastjerdi**

Internet of things: An overview

- Any ‘thing’ that has the ability to connect to the internet and collect and share information and data is a part of **Internet of Things(IoT)**
- **IoT** describes the network of physical objects (“things”) that are embedded with sensors, software and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet.

..Internet of things: An overview

- The 'things' that makeup the IoT can be anything from a wearable fitness trackers to an autonomous vehicle .
- These devices must have the following components:
  - Sensors
  - Actuators
  - IoT gateways
  - The cloud
  - User Interface

...Internet of things: An overview

- Initially **Radio Frequency identification(RFID)** used to be the dominant technology behind IoT development, then **WSN(wireless sensor network) and Bluetooth enabled devices** augmented the mainstream adoption of the IoT trend.
- The IoT creates an intelligent, invisible network fabric, that can be sensed, controlled and programmed.
- IoT enabled products employ embedded technology that allows them to communicate ,directly or indirectly, with each other or the internet.

History

- The history of IoT starts with ARPANET, the first connected network
- In **1982** , a graduate student from Carnegie Mellon University developed a system to monitor the status of **coke vending machine.**
- In **1990** , John Romkey developed **a toaster** that could be turned on and off over the internet – toaster wired to the computer(as no WiFi then!)
- **Kevin Ashton** in 1999, coined the term IoT.

...History

- In 1999, Bill Joy (founder of Sun Microsystem) envisioned **device to device** communication as part of his “six webs” framework(near web, here web, far web, weired web,B2B,**D2D**) at the World Economic Forum.

IoT Evolution represented by Three Major Phases:

- Connect the unconnected

  Connecting the legacy devices and uniting them across the network.

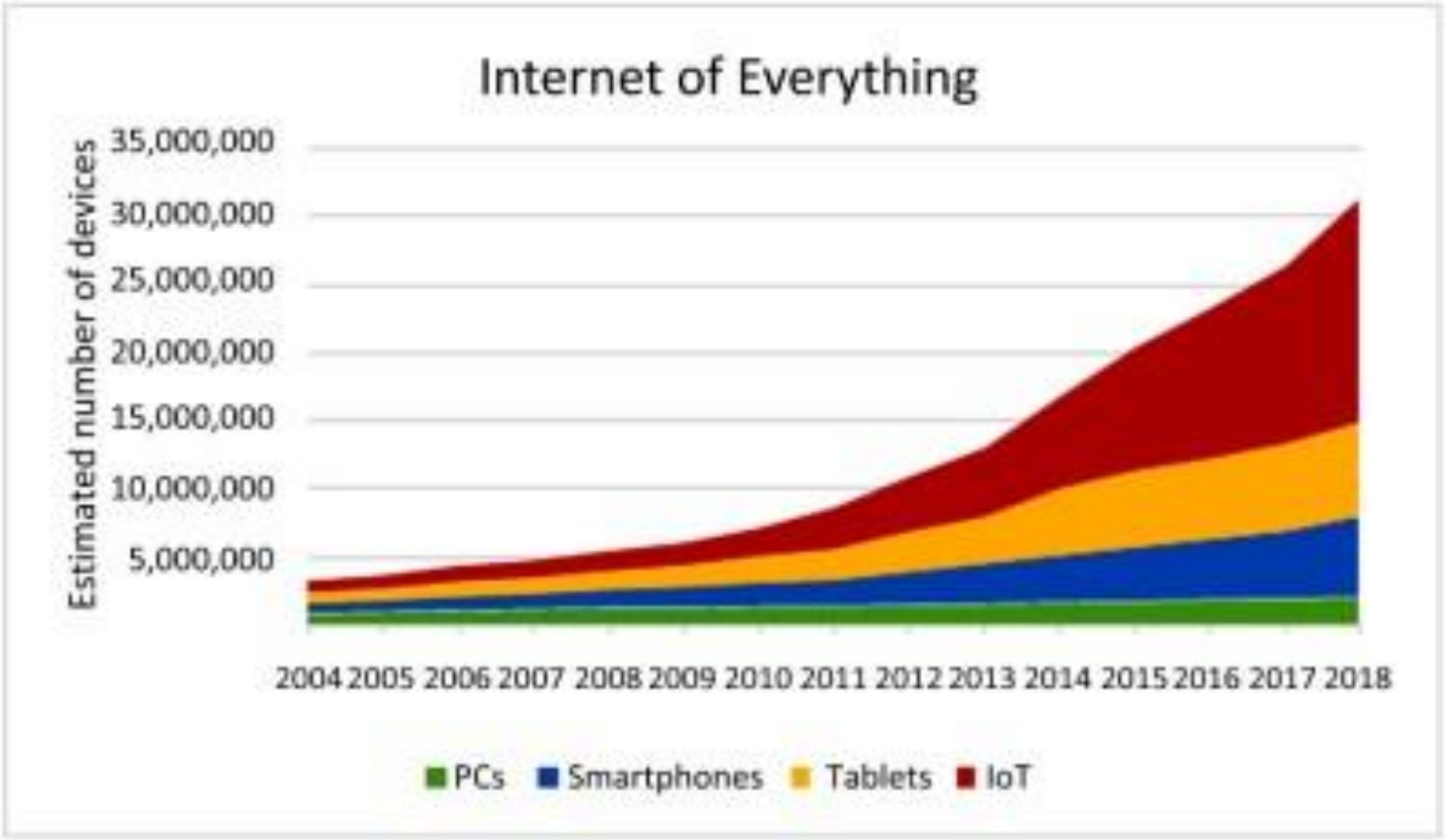
- Create smart and connected things

  Devices will need to be able to take in data, analyze the data and send the key information to the server.

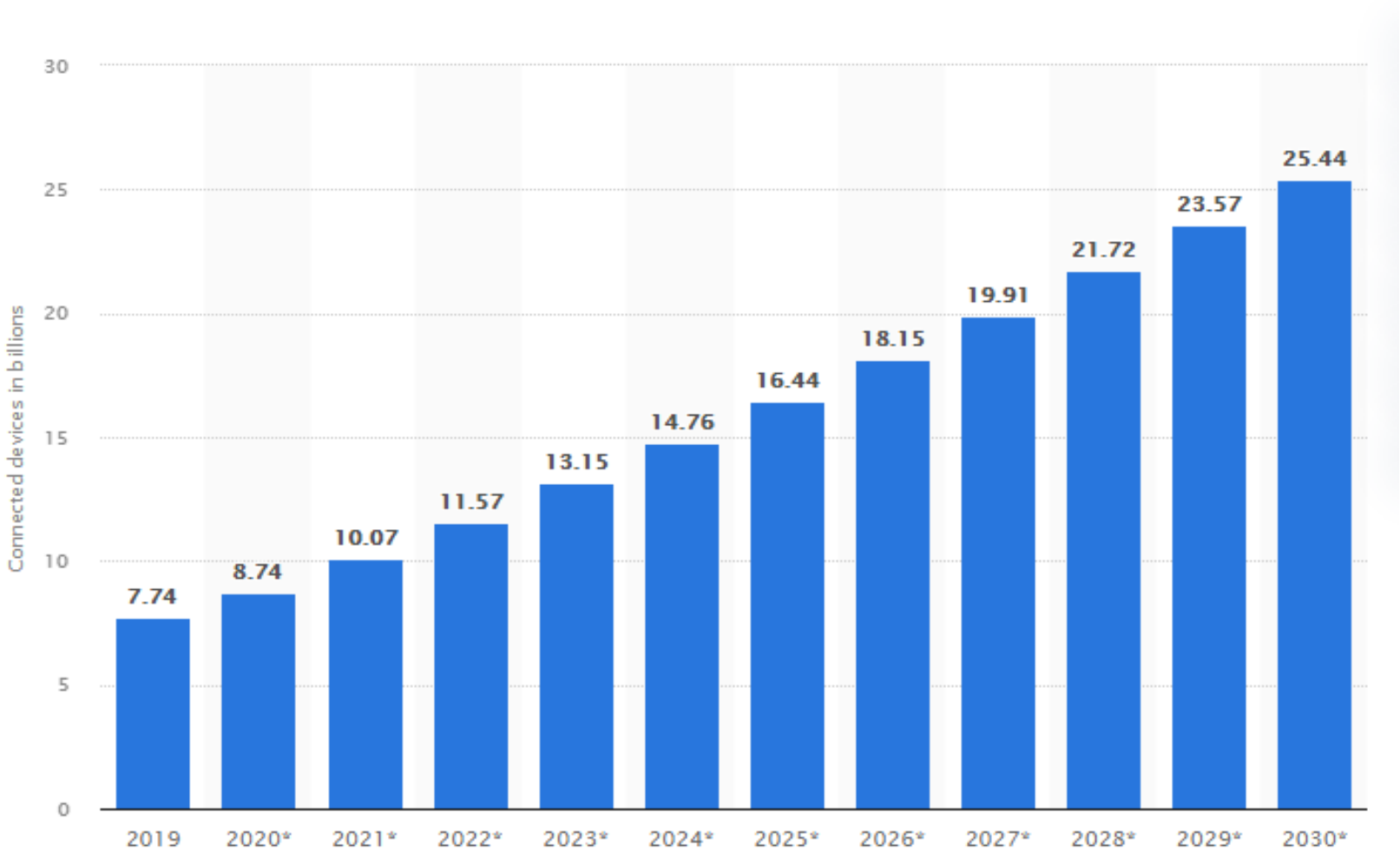
- Build a software defined autonomous world

  'Things' will become smart enough that they can start to make decisions on their own.

IoT Trend and Forecast



Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2030 :



IoT applications

- There are numerous IoT applications , which can be created to be specfic to almost every industry.
- IoT applications are using **Artificial Intelligence** and **Machine Learning** to add intelligence to devices.
- Few applications are,

➢ Smart Home

➢ Health Care

➢ Traffic Surveillance

Smart Home Applications

Smart Home devices collect and share information with one another in an integrated platform and automate their actions based on the owner's preferences**.**

- **Smart Thermostats**: which monitor and control home temperatures to the comfort of owner.
- **Smart lighting**: lighting adjusts themselves based on the user preferences as well as external lighting.
- **Smart kitchen** : can check for smoke and carbon monoxide or temperature/humidity levels- special programs monitor if the users have enough products in the refrigerator, calculate nutritional value of the meals.
- **Security systems**: With special sensors, controllers can automatically lock the door, turn off electronic devices- users can check their home state remotely through app.

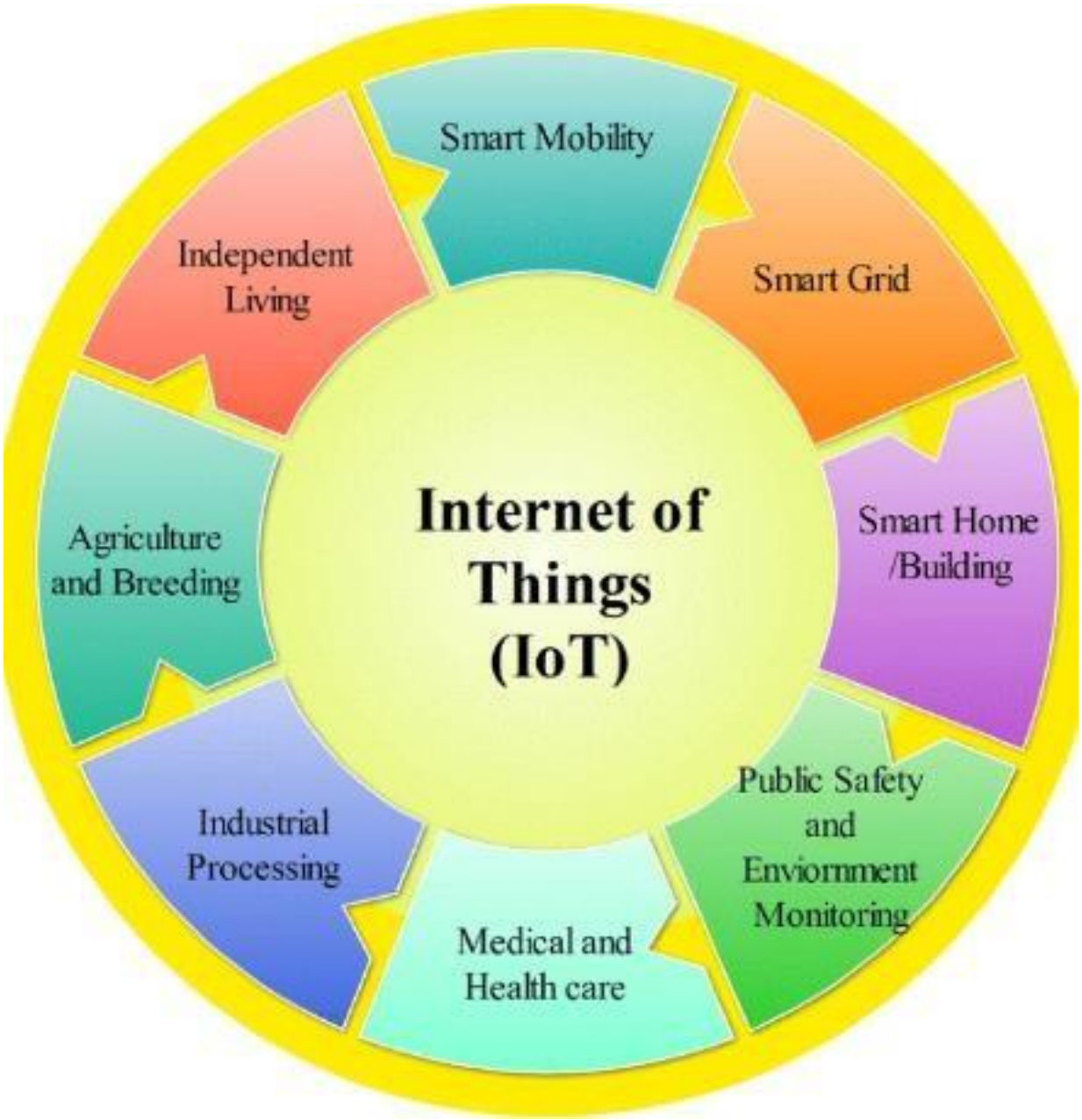
Health care applications

- **IoT for patients**: Devices in the form of wearable devices like fitness bands, blood pressure watches, ECG headbands etc.
- **IoT for physicians**: Data collected from the IoT devices can help physicians identify the best treatment process for the patients.
- **IoT for hospitals**: IoT devices tagged with the sensors are used for tracking real time location of medical equipments- Deployment of medical staff at different locations can also be analyzed real time.
- **IoT for health insurance companies**: The insurance companies can leverage data captured through health monitoring devices, which enable them to detect fraud claims.

Traffic surveillance

- **Smart traffic control**
- **Vehicle Control**
- **Smart parking**
- **Connected Car**
- **Drowsiness alerts**
- **Electronic Toll Collection System**
- **Logistics and fleet management**

and many others…


Smart Mobility
Smart Grid
Smart Home /Building
Public Safety and Enviornment Monitoring
Medical and Health care
Industrial Processing
Agriculture and Breeding
Independent Living
Internet of Things (IoT)

IoT Data Management and Analytics

- **Special considerations are required to process huge amounts of data originating from, and circulating in, a distributed and heterogeneous environment.**
- **Big Data related procedures, such as data acquisition, filtering, transmission, and analysis have to be updated to match the requirements of IoT .**
- **Big Data is characterized by 3Vs:**
  - **velocity,**
  - **volume, and**
  - **variety**

IoT Data Management and Analytics

Focusing on either an individual or a combination of these three Big Data dimensions has led to the introduction of different data-processing approaches:

Two methods used for data analysis are:

- Batch Processing
- Stream Processing

Applications utilize pattern detection and data-mining techniques to extract knowledge and make smarter decisions - but the centralized nature of these algorithms, affects their performance and makes them unsuitable for IoT environments that are meant to be geographically distributed and heterogeneous.

IoT and the Cloud

- Cloud computing can be used to analyze data generated by IoT objects in batch or stream format.
- A **pay-as-you-go model** adopted by all cloud providers has reduced the price of computing, data storage, and data analysis, creating a streamlined process for building IoT applications.
- With cloud's elasticity, distributed Stream Processing Engines (SPEs) can implement important features such as **fault-tolerance** and **auto scaling** for bursty workloads.

IoT and the Cloud

- Most of the current design approaches for integrating cloud with IoT are based on a three-tier architecture:
  - **Bottom layer** consists of **IoT devices,**
  - **Middle layer** is the **cloud provider**
  - **Top layer** hosts different **applications** and high-level protocols.
- However, using this approach to design and integrate cloud computing with an IoT middleware limits the practicality and full utilization of cloud computing in scenarios (where, minimizing end-to-end delay is the goal . Eg: Online game streaming)

Real-time analytics in IoT and Fog computing

- Current data-analytics approaches mainly focus on dealing with Big Data, however, processing data generated from millions of sensors and devices in real time is a challenging case.
- Real-time processing requirements and the increase in computational power of edge devices such as routers, switches, and access points lead to the emergence of the **Edge Computing paradigm**.
- The Edge layer **contains the devices that are in closer vicinity to the end user** than the application servers, and can include smart phones, smart TVs, network routers etc.

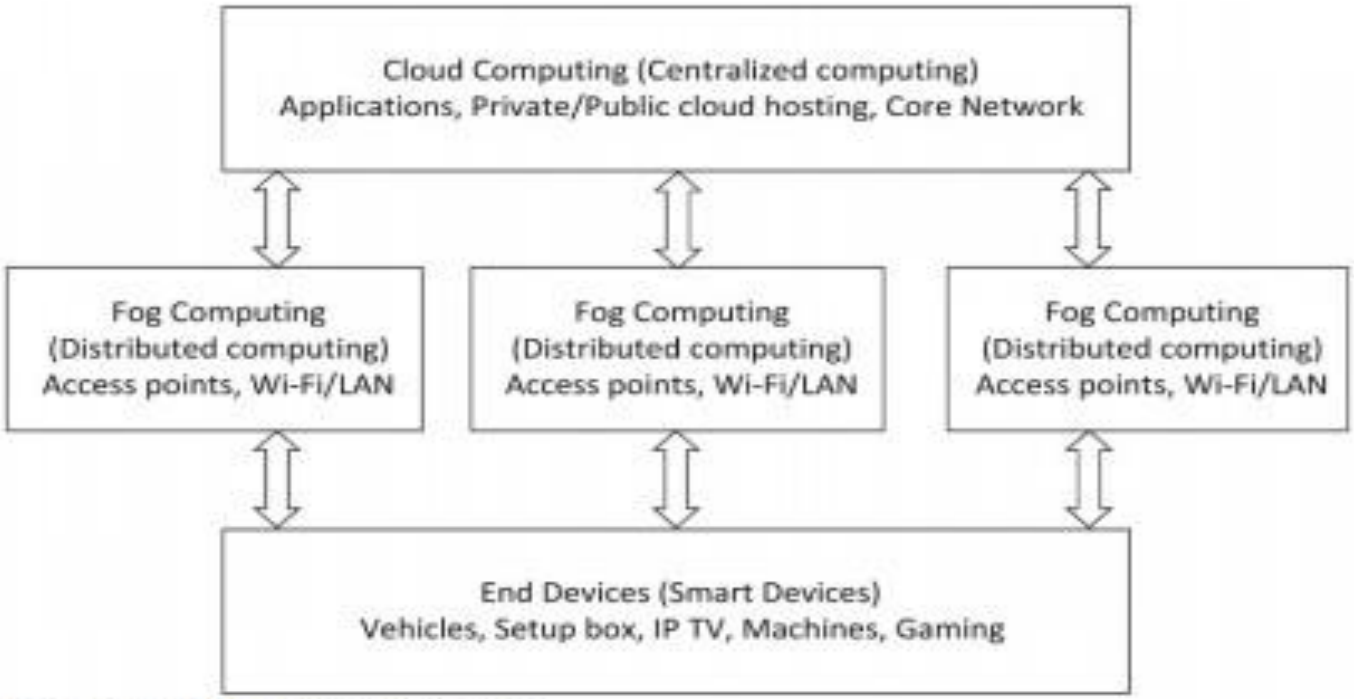
…Real-time analytics in IoT and Fog computing

- **Fog Computing** is an **extension of cloud computing** that aims to keep the same features of Cloud, such as networking, computation, virtualization, and storage, but also meets the requirements of applications that demand low latency, specific QoS requirements, Service Level Agreement (SLA) considerations, or any combination of these .
- These extensions can ease application development for **Mobile applications**, **Geo-distributed applications** such as WSN, and large-scale **systems used for monitoring and controlling other systems** such as surveillance camera networks

-

Cloud Versus Fog

	Fog	Cloud
Response time	Low	High
Availability	Low	High
Security level	Medium to hard	Easy to medium
Service focus	Edge devices	Network/enterprise core services
Cost for each device	Low	High
Dominant architecture	Distributed	Central/distributed
Main content generator—consumer	Smart devices—humans and devices	Humans—end devices



Typical Fog Computing Architecture

Open IoT Architecture for IoT/Cloud Convergence

The main elements of Open IoT software architecture are:

- ***The Sensor Middleware***

- 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 Open IoT platform and the physical world, as it enables the access to information stemming from the real world.
- It facilitates the interface to a variety of physical and virtual sensors, data streams from other IoT platforms and social networks.

Open IoT Architecture for IoT/Cloud Convergence

❖ *The Cloud Computing Infrastructure:*

- 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.
- The **cloud infrastructure** could be either a **public** infrastructure (Amazon Elastic Compute Cloud (EC2)) or a **private** infrastructure (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).

Open IoT Architecture for IoT/Cloud Convergence

❖The Directory Service

- **Stores information about all the sensors that are available in the Open IoT platform.**
- **It also provides the means for registering sensors with the directory, as well as for the discovery of sensors.**
- **The IoT/cloud architecture specifies the use of semantically**
  - **annotated descriptions of sensors as part of its directory service.**
- **The Directory Service is deployed within the cloud infrastructure, thereby providing the means for accessing sensor data and metadata residing in the cloud.**

Open IoT Architecture for IoT/Cloud Convergence

❖ The Global Scheduler:

- 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 discovering the sensors that can contribute to its fulfilment.
- It also selects the resources(sensors) that will support the service deployment, while also performing the relevant reservations of resources.
- This component enables the scheduling of all IoT services.

Open IoT Architecture for IoT/Cloud Convergence

❖ The Local Scheduler component

- It is executed at the level of the Sensor Middleware, and **ensures the optimized access to the resources managed by sensor middleware instances**, whereas
- the Global Scheduler regulates the access to the resources of the Open IoT platform ;
- 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

❖ The Service Delivery and Utility Manager:

- It performs a dual role,

➢ It **combines the data streams as indicated by service workflows within the Open IoT 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 scheduler component.

➢ It 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 "pay-as-you-go" computing paradigm

Open IoT Architecture for IoT/Cloud Convergence

❖ The Request Definition tool:

- It enables the specification of service requests to the Open IoT 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)

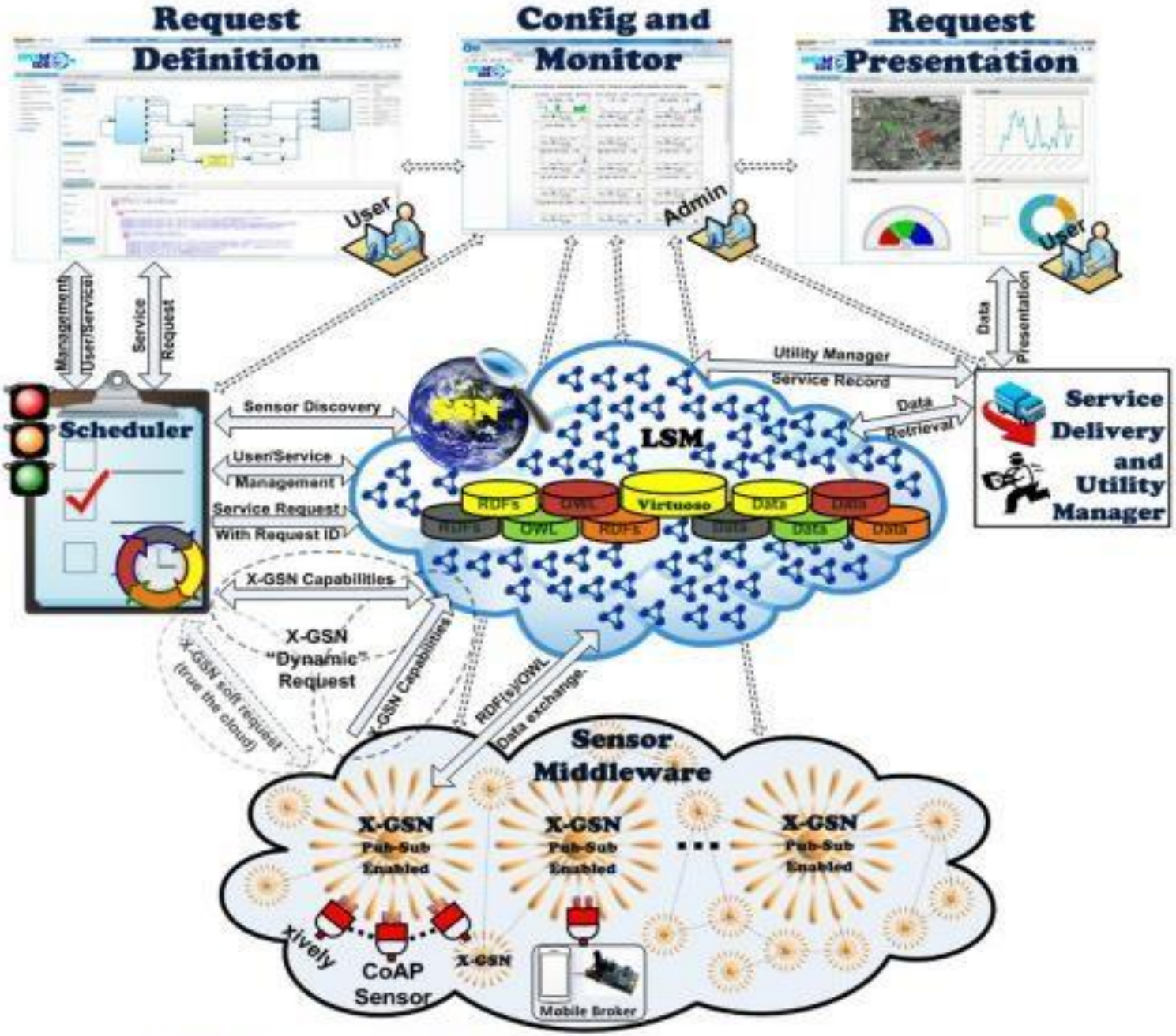
❖ The Request Presentation component:

- It 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.

Open IoT Architecture for IoT/Cloud Convergence

❖ The Configuration and Monitoring Component:

- It enables management and configuration functionalities over the sensors.
- It enables IoT services that are deployed within the platform.
- This component is also supported by a GUI.



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/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 execution of the service by end users** (based on the handle of the target service) and the visualization of the results.

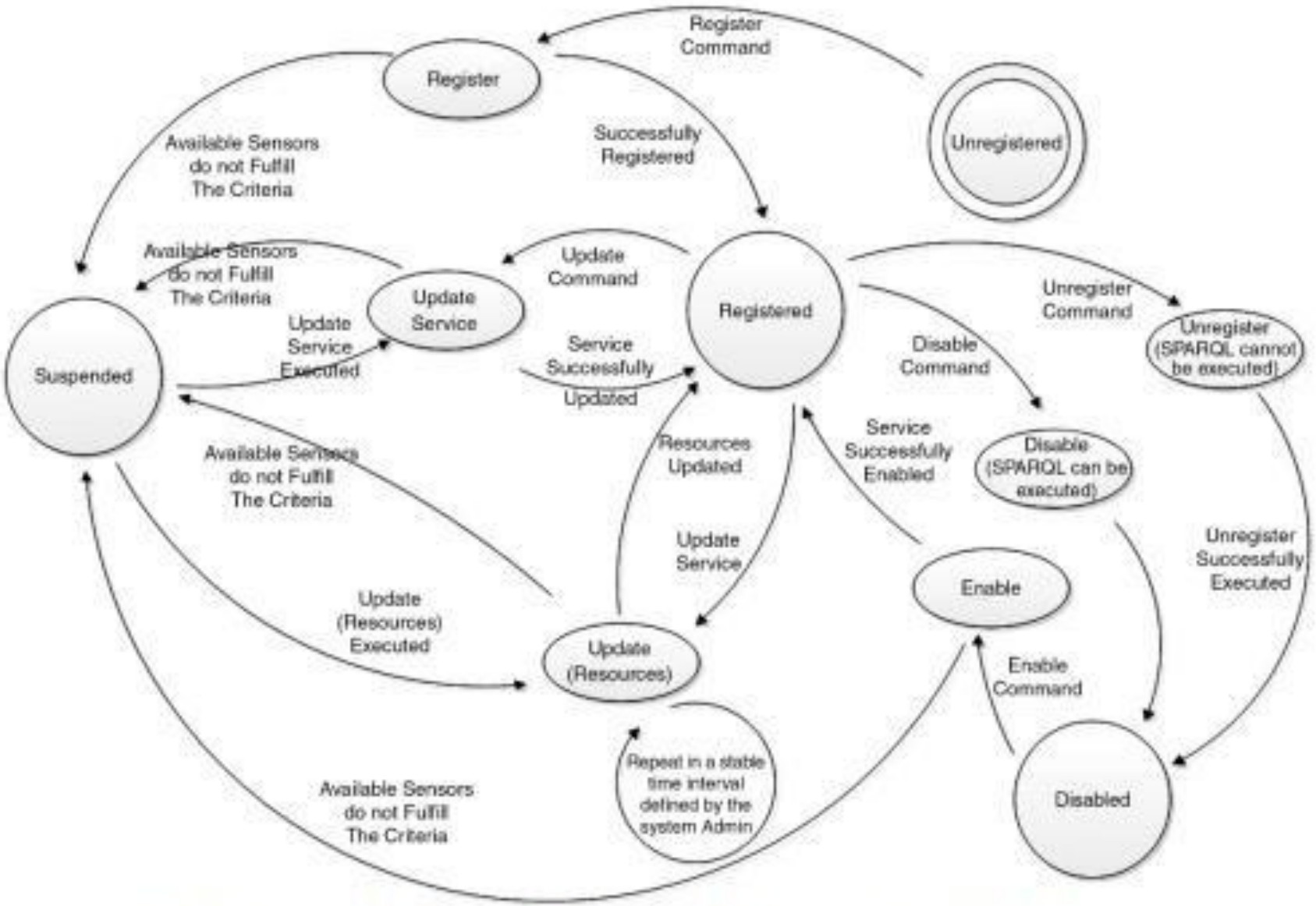
Scheduling Process and IoT Services Lifecycle

- **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 Supported by The Scheduler:

- **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-a unique identifier (ServiceID) is assigned to the requested IoT service.
  - The open source 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 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 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 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).

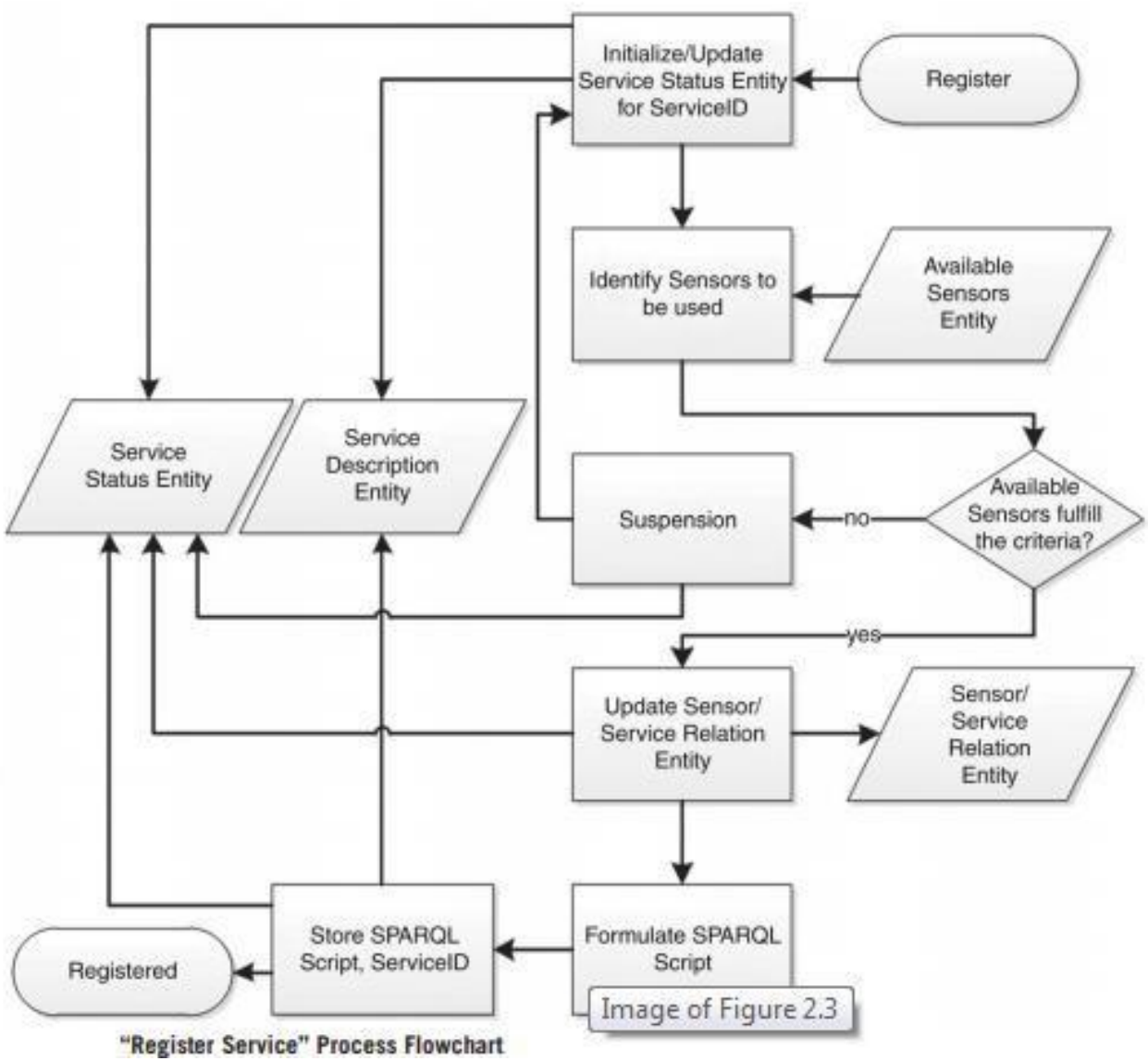
  Accordingly, it updates the IoT service metadata on the basis of the newly defined sensors that support the IoT service.

- ***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.

SPARQL is the standard query language and protocol for Linked Open Data and RDF databases.
Resource Description Framework (RDF) : RDF is a standard model for data interchange on the Web.

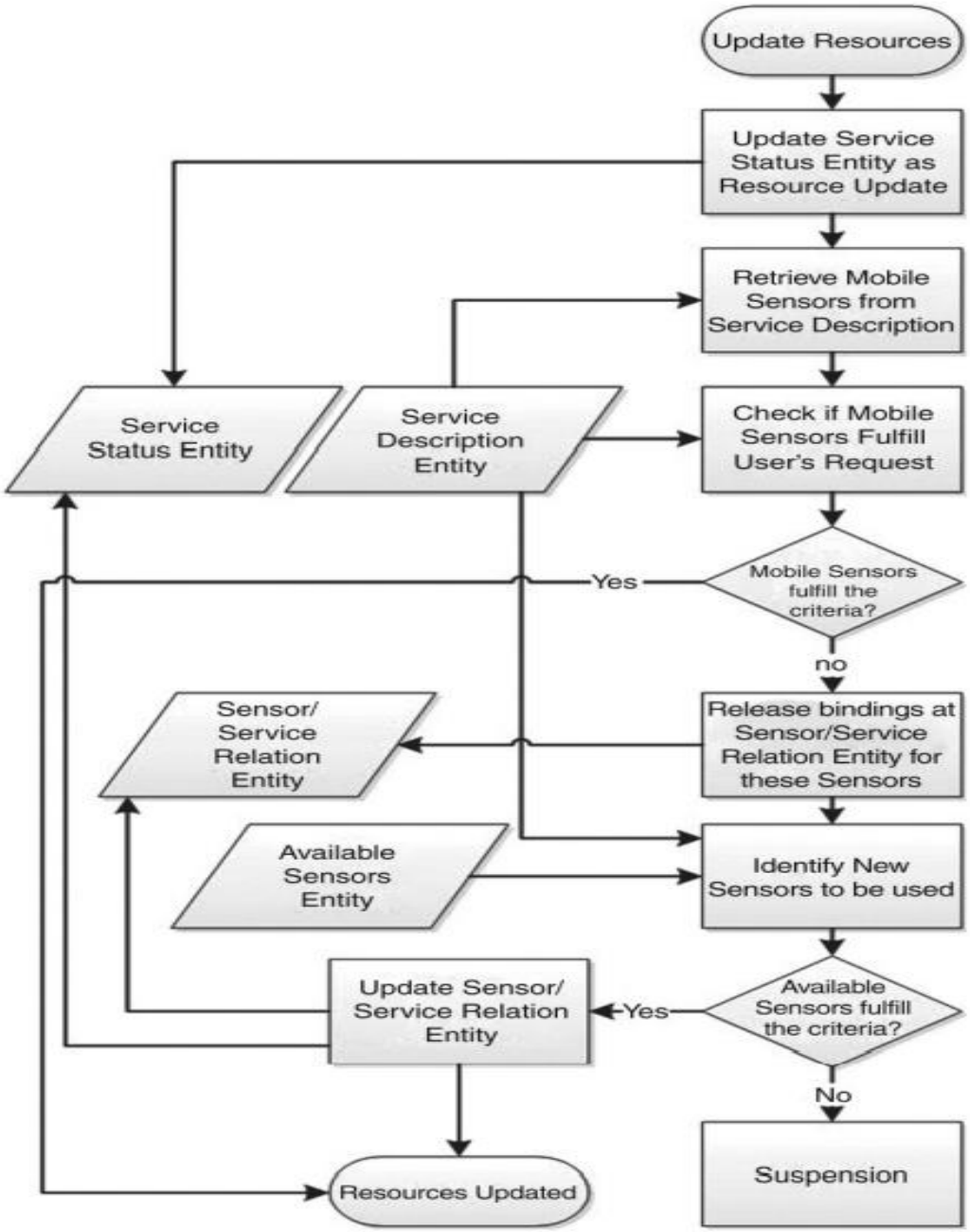
- ***Get Available Services*:** This service returns a list of registered services that are associated with a particular user(the various IoT services are registered and established by users of the platform)

“Register Service” Process Flowchart

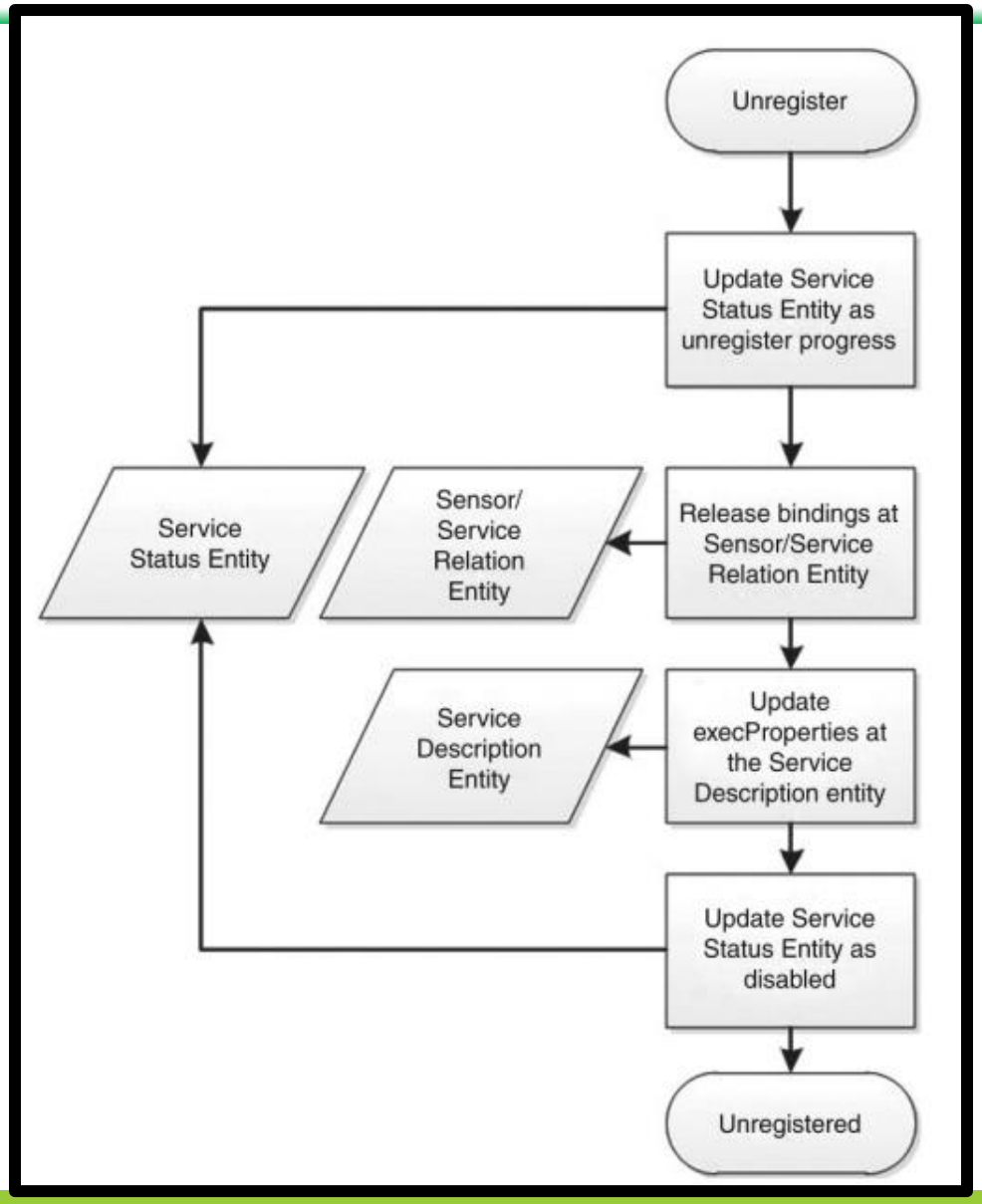


“Register Service” Process Flowchart

"Update Resources Service Flowchar



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 first optimization scheme concerns bandwidth and storage optimization through indirect control over the sensor, and falls in the broader class of in-network process approaches. 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.

SSN --> Sematic Sensor Network

➢ 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.

…Resource optimization scheme

- **X-GSN:** XGSN → Extension of the GSN middleware

  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).

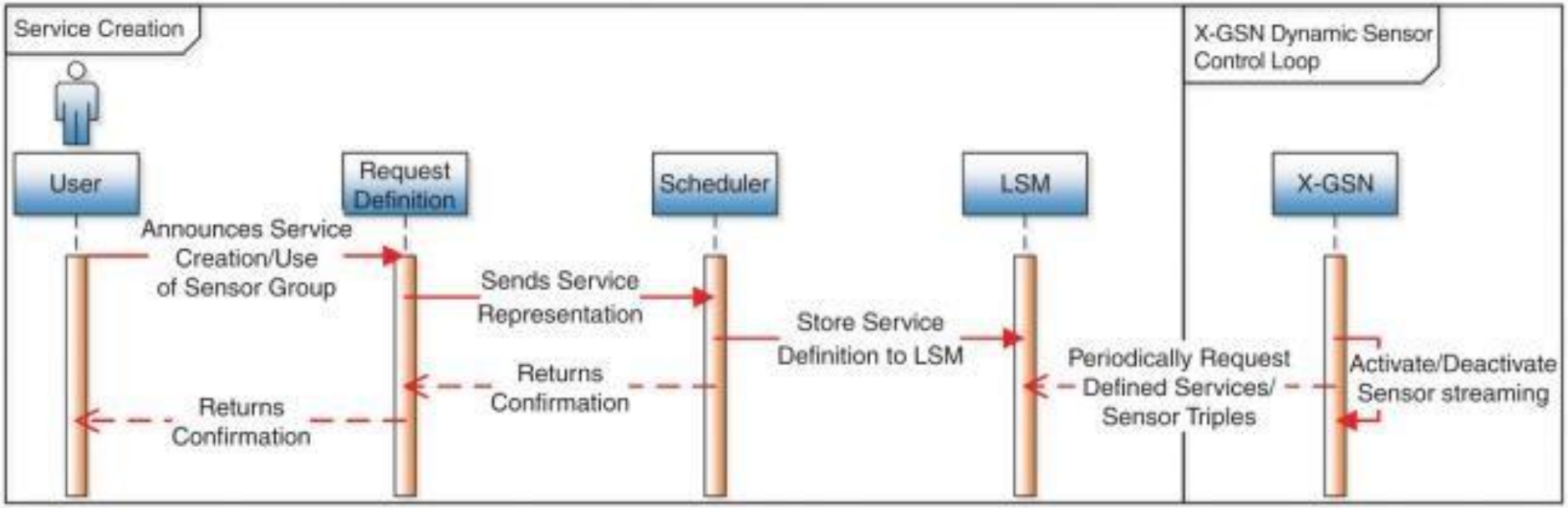
…Caching Techniques

- The caching solution that was implemented over the OpenIoT – **A small proxy layer is implemented and used to route all SPARQL queries.** Proxy Layer-->AppIn Layer Gateway
- 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).

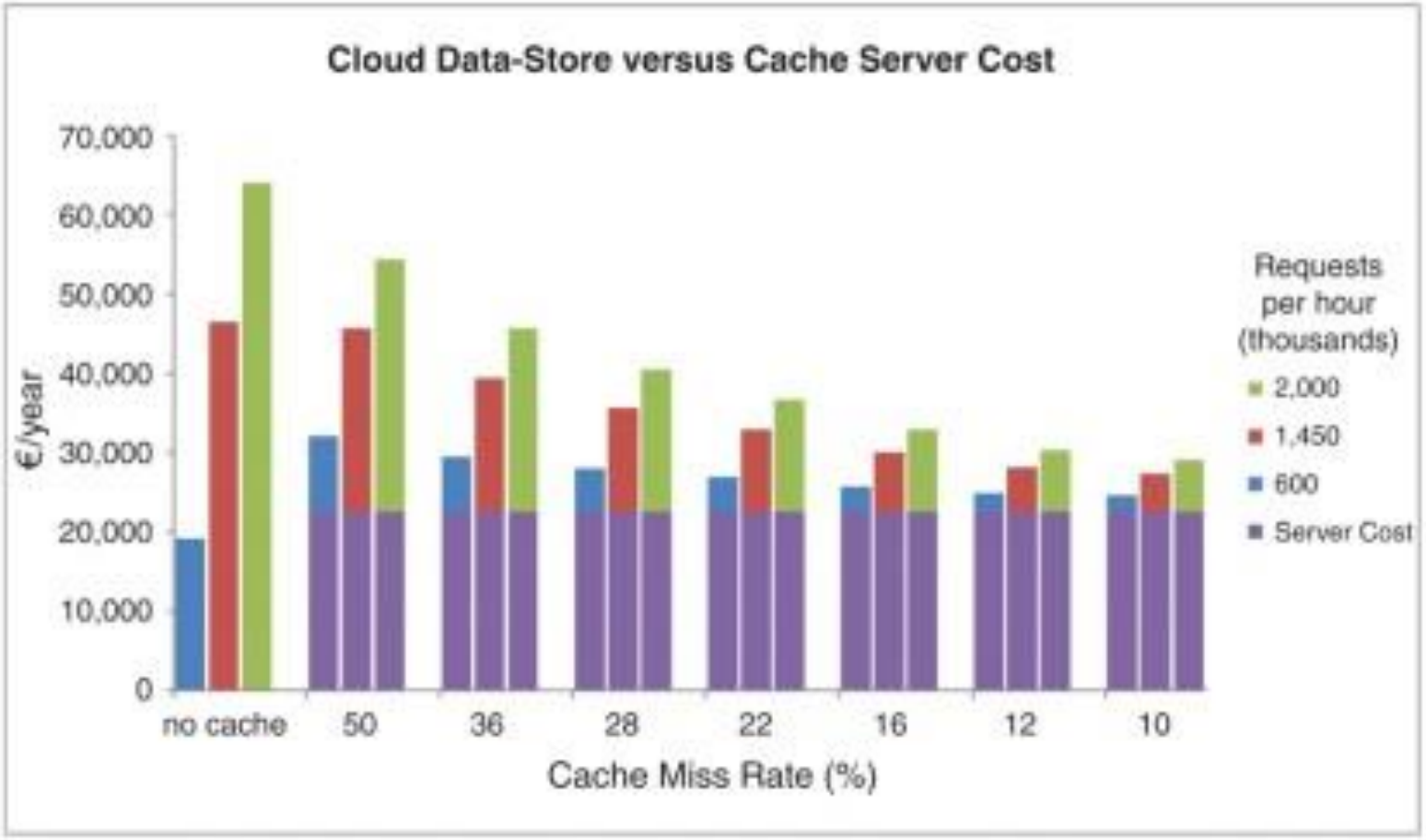
…Caching Techniques

- 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



"Service Creation" Service Flowchart



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

…Caching Techniques

Figure(illustrates the total costs)incurred for seven different scenarios of the SPARQL queries ,that correspond to different varieties of queries, which result in different miss-rates.

- **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.**
- **In the last category at 2000 Krph, that at a high number of requests it is far more efficient to use cache.**

Device/Cloud Collaboration Framework

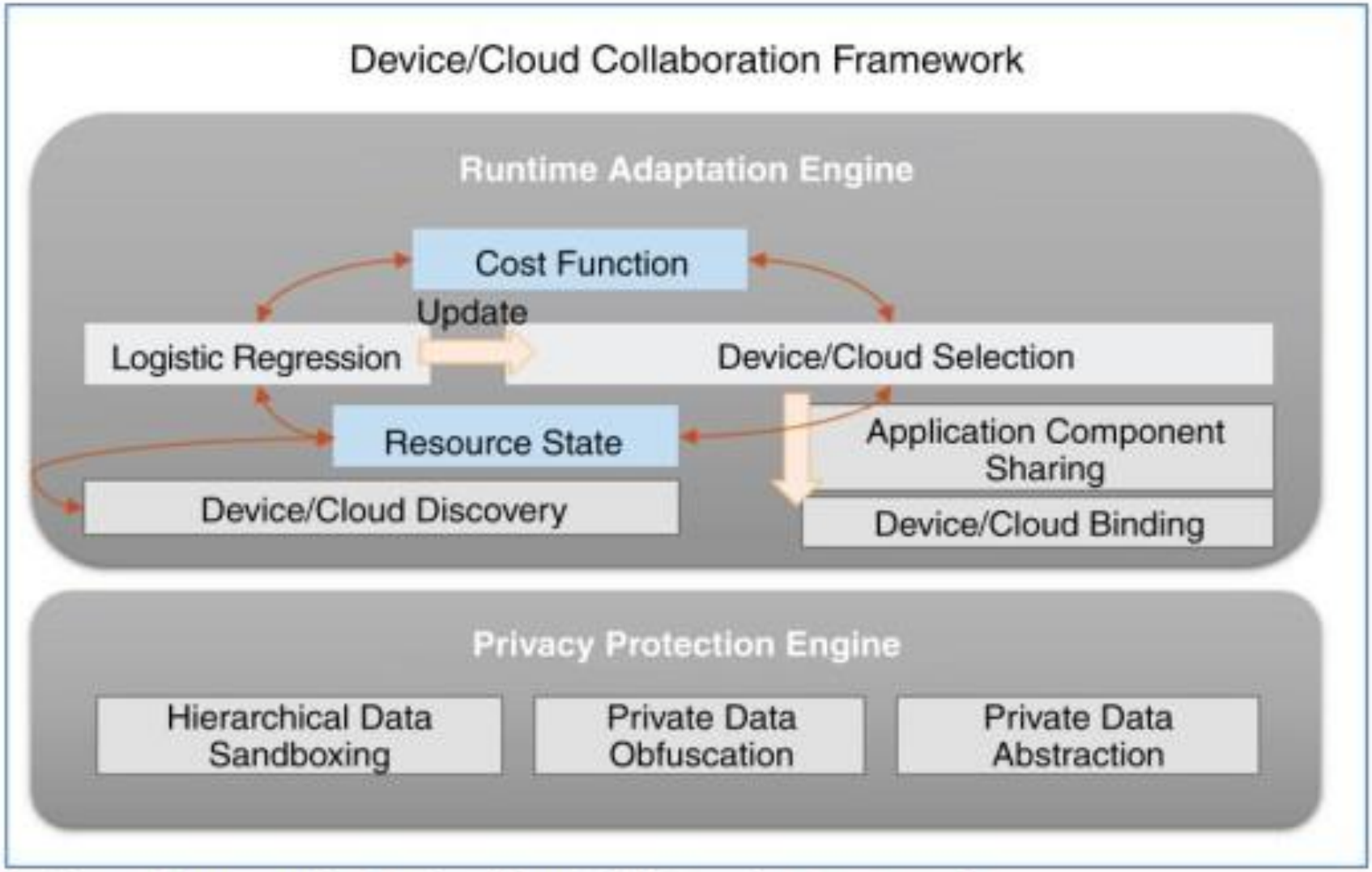
- Cloud computing is now an established computing paradigm that offers on-demand computing and storage resources to the users who cannot afford the expenditure on computing equipment and management workforce.
- We need a **cloud-computing framework** that improves both the scalability and the privacy-protection mechanism.
- At a high level, this framework leverages the **compute and storage resources** on the smart mobile devices.
- Also, this **framework enables security solutions** that protect privacy without degrading the quality of applications.
- **Device/Cloud collaboration framework** is a Framework that enables **collaboration between smart mobile devices and cloud** for a more **scalable and secure computing mechanism**.

Device/Cloud Collaboration Framework

Powerful smart mobile devices

- Smart phones nowadays have enough computing capacity to process various computing tasks.
- However, the device usage can be constrained by limited battery life and network connectivity.
- Therefore, we can consider utilizing the highly available cloud resources in addition to the device.

Device/Cloud Collaboration Framework



High-Level Layout of the Device/Cloud Collaboration Framework

Device/Cloud Collaboration Framework

- For a given a computation task, framework needs to choose the entity to execute the task.
- For eg, we are given a task of **processing a query issued over voice**, and assume that we have a lightweight mobile version of a voice-query processing engine that is embedded in a Smartphone
- This mobile engine can answer the given query without the cost of
  - transferring the voice data to the cloud over the network.
- But, it will consume the limited battery life of the device
- If the lightweight voice-query engine returns a poor result, then the user may have to issue the query redundantly to the cloud-based query processing engine with the hope of getting a better result. This may hurt the overall quality of experience (QoE).

Runtime Adaptation Engine

- So we require a **decision mechanism** that **automatically selects a better *agent* that can execute a given task.**
- **Runtime Adaptation Engine (RAE)** sits at the core of the framework.
- The RAE maintains **a list of available devices and cloud** to utilize Device/Cloud Discovery, and **monitors the state of their available resources**.
- The RAE employs a **Logistic Regression algorithm** to learn the **most cost-effective policy for distributing tasks** among devices and cloud, given the resource state.

Runtime Adaptation Engine

- Definition of the cost function is the **weighted sum of the resource state** (battery life, network, and CPU usage…)
- The policy obtained by running the Logistic Regression is enforced by the Device/Cloud Selection module that chooses the most economical compute resources, based on the expected cost-value for a given task.
- The mechanism of the RAE is **an autonomous agent**, which can be
  - deployed on each device and cloud.
- **RAEs communicate with each othe**r to transparently share the resource state **for determining the ways to distribute a given workload.**

Runtime Adaptation Engine

- The cloud-side RAE can also model its own cost function as a **weighted sum of residual CPU cycles and storage space** across the entire infrastructure.
- If the residual capacity falls below particular thresholds, the cloud may have to reject the resource-sharing request coming from the paired devices. This is because running the requested task would be too costly.
- The cloud-side RAE advises the device-side RAE to **either execute the task within the device** or simply **wait for the compute resource on the cloud to be freed up.**
- The RAEs on the devices and the cloud make decisions autonomously, without any supporting brokerage system in the middle.

Runtime Adaptation Engine

- Device-side RAE has the burden of **periodically monitoring the state of the cloud resources.**
- But ,the cloud-side RAE **does not have to monitor the resource state of the millions of paired devices**, as the cloud makes a relatively simple decision, that is, to either reject or accept a task-execution request.
- By default, this framework does not consider offloading cloud-initiated tasks to the devices.

Runtime Adaptation Engine

Device/Cloud Discovery component:

- To support task distribution among the neighbouring devices, the device/cloud collaboration framework implements discovery of devices

Application Component Sharing :

- Suppose device A wants to collaborate with its neighbouring device B which is not equipped with appropriate application components to process a given task.
- Then, device A can transfer the necessary application components to device-B through the Application Component Sharing.

Applications Of Device/Cloud Collaboration

- Device/collaboration framework can be used by the real- world intelligence applications developed specifically at Samsung Electronics.
- The selected applications offer the following functionalities:

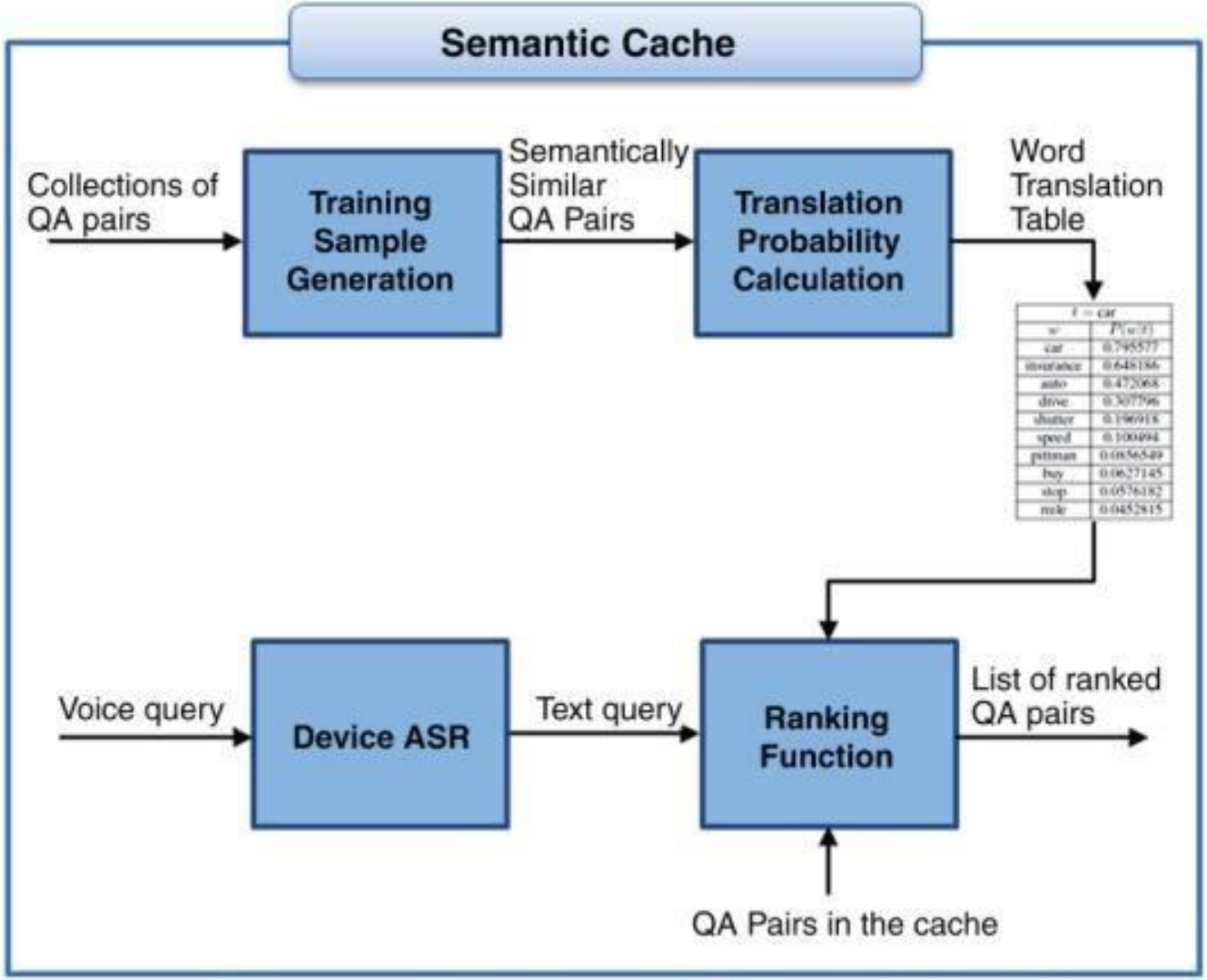
➢ *context-aware proactive suggestion*

➢ ***semantic QA caching***

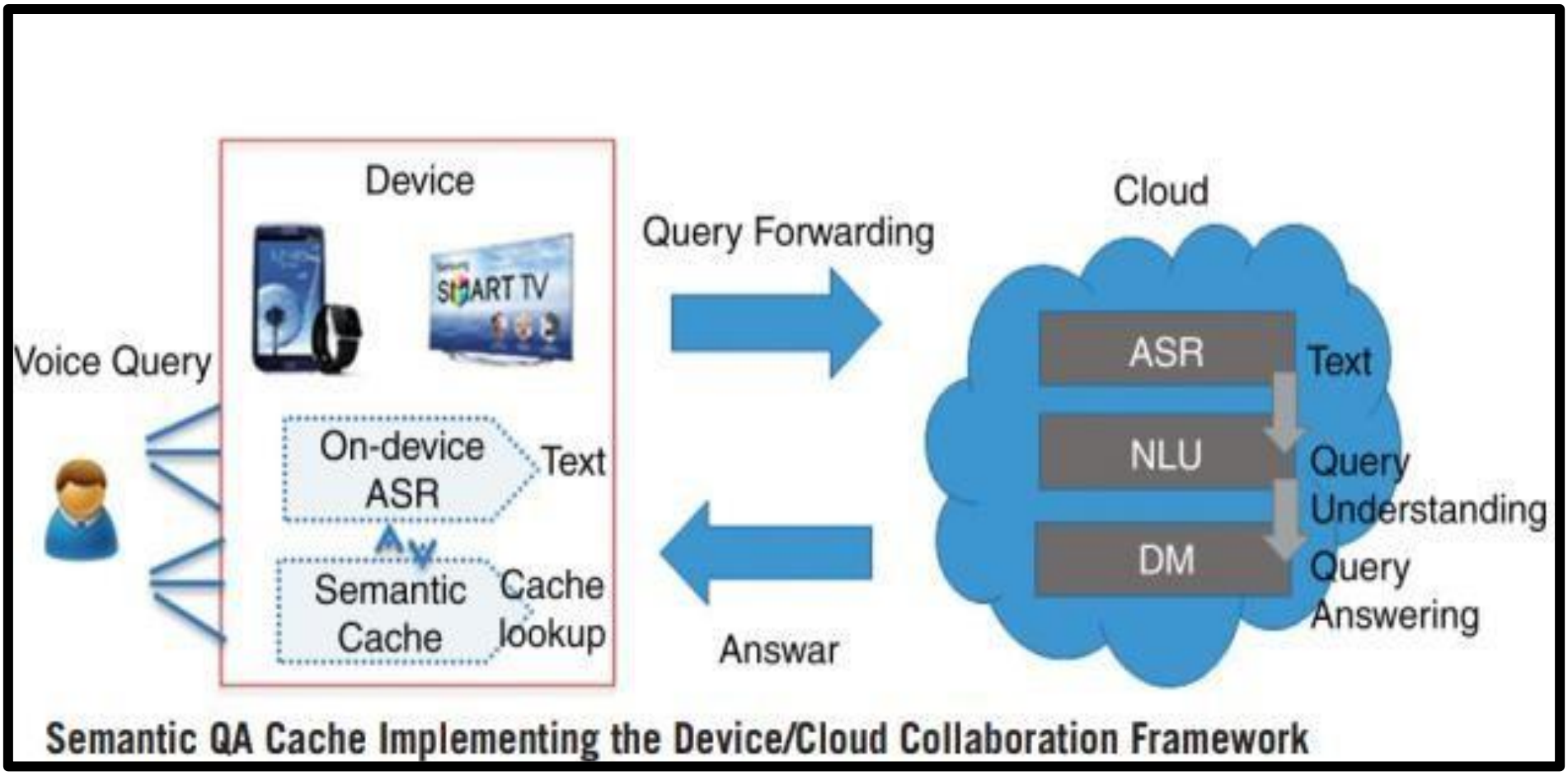
➢ *automatic image/speech recognition*

Semantic QA Caching

- Semantic QA cache is a mobile application that **retrieves answers to a given query from the cache filled with answers to the semantically similar queries issued in the past.**
- Semantic QA cache can be **useful when there is no Internet connectivity** or when the user is not in favour of transferring private queries to the cloud.
- Semantic QA cache **returns a list of similar queries and the**
  - **associated answers.**
- Semantic QA cache **constantly updates ranking function**
  - Based on the word-translation table.
  - The ranking function measures the similarity between a newly issued query and the queries measured in the past.



Illustrations of the Technique to Cluster Semantically Similar QA Pairs for Retrieving an Answer for a Newly Given Query Without Asking the QA Engine on the Cloud Side



Semantic QA Cache Implementing the Device/Cloud Collaboration Framework