UNIVERSITY OF NAMUR

INITIATION À LA DÉMARCHE SCIENTIFIQUE

IHDCB339

State of the Art: Complex Event Processing for Internet of Things

Author
Kenny WARSZAWSKI

Supervisors
Moussa AMRANI
Pierre-Yves SCHOBBENS

August 8, 2019

UNIVERSITÉ DE NAMUR

1 Introduction

The Internet of Things is omnipresent and the amount of connected devices is increasing day by day. The Internet of Things is different from a classic information system by its capability for integrating with the physical world. Those devices are used in plenty of sectors (health, industry, transport, home automation, ...) and their users can be both professionals and individuals. Nowadays, we can see the apparition of connected devices for home to facilitate our daily life. (e.g: Google Home, Nest, Philips Hue) IoT is also an interesting technology for Smart Cities use case. We can imagine a city where traffic lights are optimised with the city mobility to avoid traffic jams for example. However, an important problem arises in this type of architecture. How is it possible to handle such an important data traffic efficiently ? Indeed, if an entire city has a huge amount of connected objects, the data flow to be processed is massive. Therefore efficient data processing mechanisms need to be put in place to handle such flows.

Parler des CPS, Fraud Detection, Health, Fault Tolerance, structure de l'état de l'art, etc

2 Context

2.1 Internet of Things

The Internet of Things(IoT) is a novel paradigm that is rapidly gaining ground in the scenario of modern wireless telecommunications. The basic idea of this concept is the pervasive presence around us of a variety of things or objects – such as Radio-Frequency IDentification (RFID) tags, sensors, actuators, mobile phones, etc. – which,through unique addressing schemes, are able to interact with each other and cooperate with their neighbors to reach common goals. [2]

The phrase "Internet of Things" was mentioned the first time by Kevin Ashton, the co-founder of the Auto-ID Center at MIT, as the title of one of his presentation. For him, today computers and the internet more generally is dependent from human intervention. The information technology architecture are all designed around the hardware, software interactions, etc but an important thing is generally set aside: people. People have a limited time, attention and accuracy. Hence they are making mistakes about the real world. Thus, the data generated by humans isn't the exact representation of the reality in an Information System.

A thing can be any connected device from our daily life that is able to capture information from the physical world by sensors and send it through a network. This information can be kept by another layer able to process the data and make it meaningful by an application.

2.2 IoT architecture

The architecture of an Internet of Things solution can be implemented with various technologies and in very different infrastructures. However, a generic high level architecture is common to all IoT projects:

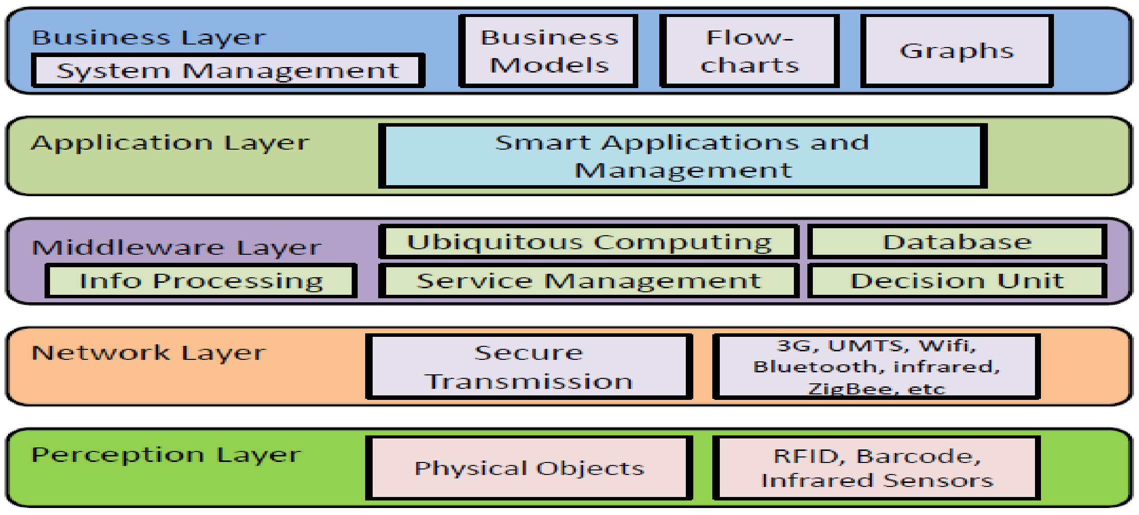

Figure 1: Generic Architecture

Perception Layer: The perception layer or device layer is the layer where physical objects will be able to capture data via sensors. These sensors are able to collect real world data. These data can be or not be processed upstream to add meta-data by the connected object itself and will be forwarded to the Network Layer.

Network Layer: The Network Layer or Transmission Layer. This layer transfers the informations from to Perception Layer to the Middleware Layer. This transfer can be accomplished by different communication technologies like Wifi, 3g, Bluetooth, etc.

Middleware Layer: The Middleware Layer is the common layer for all physical objects of an IoT architecture. The middleware is responsible to take automatic decisions based on informations coming for the devices and store the results in a database.

Application and Business Layer: The Application Layer is where smart applications are running to produce complex processes. These applications will consume data from connected objects previously processed by the middleware to act effectively on a specific situation. These application can transmit actions to execute on a particular type of service. The Business Layer is important for the global management of the system. This one is rather a layer of global monitoring which produces graphs and statistics at the business level.

2.3 Data Flow

The previous architecture can be simplified to highlight the layer of architecture on which this state of the art will focus.

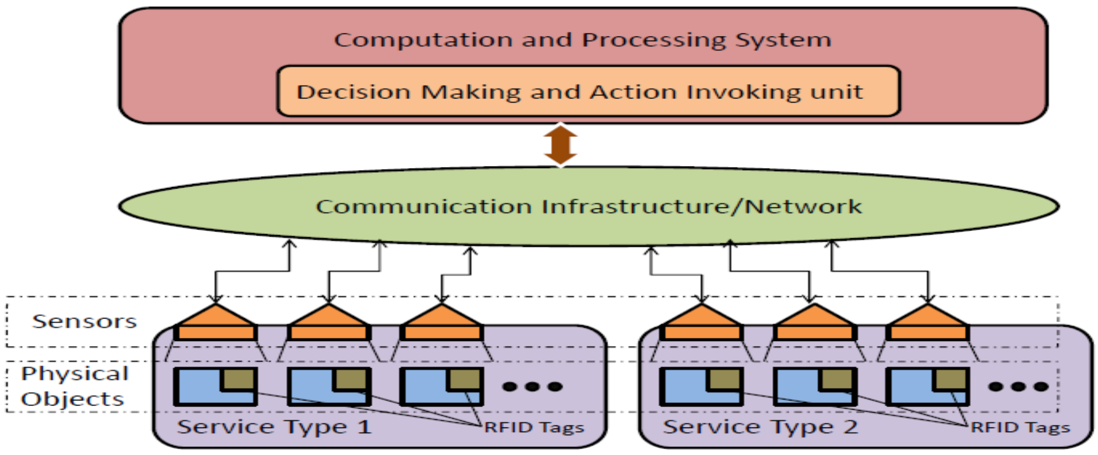
Computation and Processing System
Decision Making and Action Invoking unit
Communication Infrastructure/Network
Sensors
Physical Objects
Service Type 1
RFID Tags
Service Type 2
RFID Tags

Figure 2: Simple IoT Architecture

This state of the art is focusing on the "Computation and Processing System" layer because at that stage, the data processing is the most critical and the data flow is the highest. Obviously, the Communication Layer must be robust enough to transmit the data efficiently but this topic will not be tackled in this article.

3 Complex Event Processing

3.1 Definition

Complex Event Processing (CEP) is a set of methods and techniques for tracking and analysing real-time streams of informations and detecting patterns or correlations of unrelated data (complex events) that are of interest to a particular business. [6] The complex event processing is the engine of real-time applications that needs to have real-time informations from an environment to respond as fast as possible. A standard Request/Reply with synchronous processes cannot correlate different type of events and respond in a timely manner. Complex Event Processing mechanisms are often used in Event Driven architectures. This architecture fits with the Internet of Things needs. This kind of architecture is driven by events sent by connected objects. Then, the services at the application layer are consuming those events.

3.2 Event Correlation

The events coming directly from the Thing's sensors are call raw events. Those events are the trigger of one or multiple process in an IoT solution. The raw events are the events coming from the real world. It may represent a simple real-world event or a complex real-world event. Those raw events often need to be pre-processed before integrating a more sophisticated system. Without the pre-processing it would be difficult for a system to understand the information correctly and correlate it with other events. A strict semantic between different type of events have to be put in place for the whole IoT system. That's why a complete ontology should structure all the domain application. Generally, two types of events can occur in the system: Internal and External. The internal events are the events sent by the application layer. The external event are the events sent by the "physical" sensors.

3.3 CEP Engine

The CEP engine is consuming the events sent by the physical objects on the Communication Layer. This engine can define a bunch of rules where each event received will be evaluated. A rule represents a complex condition that can be based on multiple events and on a time window. For example, if all connected cameras in a house are sending that there aren't any move in the house in a time window of 60 seconds, an event saying that nobody is in the house will be sent. Then, some specific actions can be taken by connected devices in the house. Basically, a Complex Event Processing Engine correlates multiple type of events based on the time. Then, if the pattern defined in the rule is matching with the ingested events, a complex event is sent. A complex event is an event that summarizes, represents, or denotes a set of other events[**glossary**]. The complex events generated can themselves triggers another CEP rule or an application process.

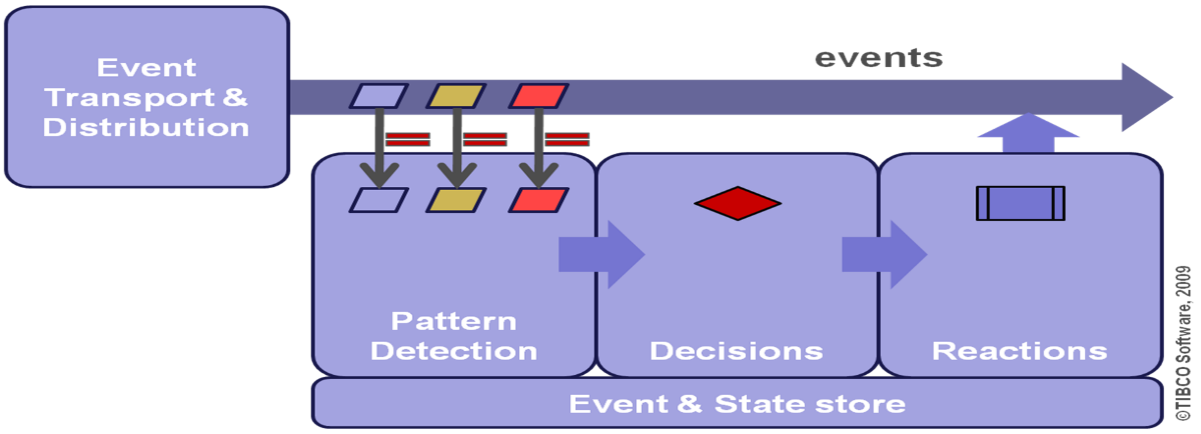

Figure 3: CEP Pattern Decision Reaction

The rules are able to correlate different type of events only if their structure are known. If an event is unrecognisable by the pattern, this event can be ignored or a different kind of action can be put in place. It is possible to send a warning event that something unexpected by the system happened for example.

The technologies implementing CEP concepts are querying Time Series Databases where each event received are stored. Thoses databases are optimised for queries based on a time window. In a lot of CEP framework like Apache Flink, those CEP pattern matching rules are describe seemly an SQL Query and uses Event Pattern Languages.

3.4 Event Pattern Languages

An Event Pattern Language (EPL) is a language used by CEP engines in order to describe the relations between events that match a specific pattern. The syntax of EPL's are quite similar to SQL queries. The Example 1 is retrieving dates where the temperature was bigger than 30 degrees Celsius.

Example 1:

```
SELECT Thermometer.date
FROM DataSource
WHERE temperature > 30
```

A plenty of primitive operators exists in Event Pattern Languages to describe the rule's behaviour. The following operators are coming from ThingML framework but are available in most of CEP frameworks:

Selection: Filters relevant events based on the values of their attributes.[3] For example, we can select all Air Pressure events between 101300 Pa and 101400 Pa.
Projection: Extracts or transforms a subset of attributes of the events.[3]
Window: Defines which portions of the input events to be considered for detecting pattern.[3] For example, a rule can concern the last 30 seconds events.
Conjunction: Consider the occurrences of two or more events.[3] This operator is basically a logical 'AND' operator.
Disjunction: Consider the occurrences of either one ore more events in a predefined set.[3] This operator is basically a logical 'OR' operator.
Sequence: Introduces ordering relations among events of a pattern which is satisfied when all the events have been detected in the specified order.[3]
Repetition: Considers a number of occurrences of a particular event.[3]
Aggregation: Introduces constraints involving some aggregated attribute values.[3] For example, the average temperature of all Weather events is an aggregation.
Negation: Prescribes the absence of certain events.[3] This operator is basically a logical 'NOT' operator.

3.5 Event-Condition-Action Pattern

As seen in section 3.3, the CEP Engine is composed of events to detect, a set of rules to execute and actions to perform. Basically, this CEP approach is called Event-Condition-Action and separate the operations in three stages : Event processing, Condition evaluation and Action execution. This pattern is often used in the most popular CEP Frameworks (e.g.Apache Flink, Esper, Google Dataflow and Apache Heron). The Event processing stage is itself divided into three parts : Event detection, Event subscription and Event processing. The aforementioned stage is responsible to detect events, subscribe and process events. The condition stage is the step where rules are defined by Event Pattern Languages. The Rule Engine is composed of a fact base and rule base. The fact base represents the current state of all facts defined in the system. A fact is a situation that can occur in the system. For example, if a detection event is sent by a sensor in a bedroom, the fact can be "Someone is in the Bob's bedroom". Rules consist of an ECA based pattern: on <event> if <condition> then <action>. Whenever an event occurs, the rule scanner checks all relevant rules to determine any consequent actions that need to be initiated. [**IoT Architecture based on services and Events**]

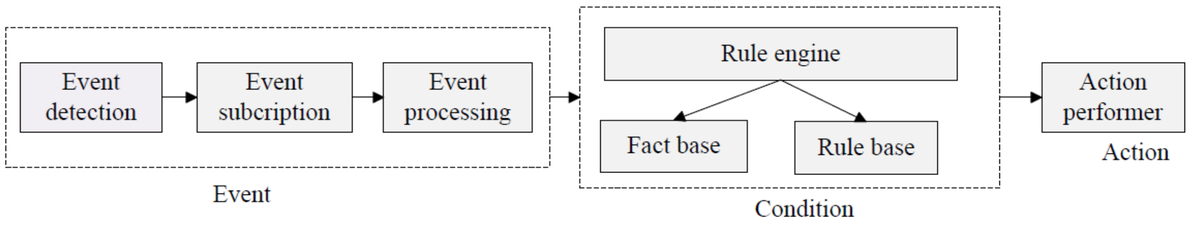

Figure 4: Event-Condition-Action

3.6 Database and Data Stream Management Systems

Real-time applications are consuming the data in a different way comparing to a traditional system. A traditional information system using a Database Management Systems (DBMS), the information stored in the database related to an specific entity is a "static" data persisted. The information stored in an aggregation of the reality. Only the current state of the information is relevant for the system. This type of Data management doesn't fit all real-time applications' needs. A real-time application needs to correlate each events coming into the system. An IoT solution is sending too much data into the system. It's not possible for a traditional database to store, query and respond in a timely manner.

The Data Stream Management Systems (DSMS) are designed to handle potentially infinite and fast changing event streams. Obviously, it is not possible to query the all history of events that came from the beginning of the solution and respond in milliseconds. Different approaches exist to limit the amount of data processed. Roughly, compression techniques exists to summarize a set of data already queried. Another technique consists of setting a time window to pick a portion of data in the time.

4 Computing Levels

The main preoccupations of a real-time solution in IoT is to have the lowest latency between all devices. To process all data coming from sensors, CEP engines can help reducing the processing time. The CEP engines can be placed at different level of the IoT architecture. The place where Complex Event Processing is used depends of the business needs.

4.1 Cloud Computing

The Cloud Computing places the computing complexity at the cloud level. All the data coming from the sensors have to transit by the connected object, sometimes by a middleware and then it can be consumed by a CEP Engine in the cloud. The Cloud Computing is common for near real-time applications where a little bit of latency is not critical.

This solution is optimal for the near real-time cloud monitoring because all events coming from the devices are sent to the cloud. Then a precise metrics history can be retrieved. Obviously all data persisted have a cost. This type of solution is extremely demanding in storage space if all the history is persisted.

Of course the Cloud Computing can be coupled with other levels of computing and it's often the case. The Cloud Computing can be placed on the top of a multi site IoT solutions where millions of data are coming from thousands of connected objects in the world. Then this solution

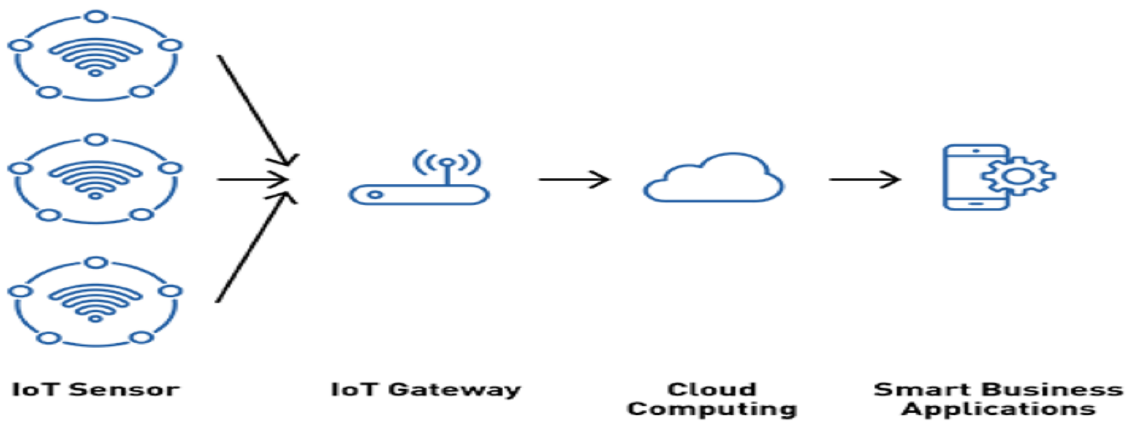

Figure 5: Cloud Computing

can reflect a High Level overview of a global IoT system and available all over the world.

4.2 Fog Computing

The Fog Computing places the computing intelligence at the local network level. Typically the complexity is put between the Cloud and the IoT sensors. The devices are sending their data to a Middleware and the CEP engine is processing the data locally and send the processed events to a service running in the Cloud or on a local server. This method offers the CEP performances on a closer level where the data are created. Then the applications where latency is crucial, the Fog Computing can fit with their business needs. For example, the auto mobile or aerospace sector are more and more connected and the reactivity of their system needs to be as fast as possible. Compared to a device-to-cloud architecture, placing processing closer to the devices can reduce the latency since the physical distance is shorter and potential response time in a data center can be removed. Compared to a deviceonly architecture, latency can be reduced since computationintensive tasks that take a long time on resource-constrained sensor devices can be moved to more capable fog computing nodes. The motivation can also be to keep the latency predictable.[1]

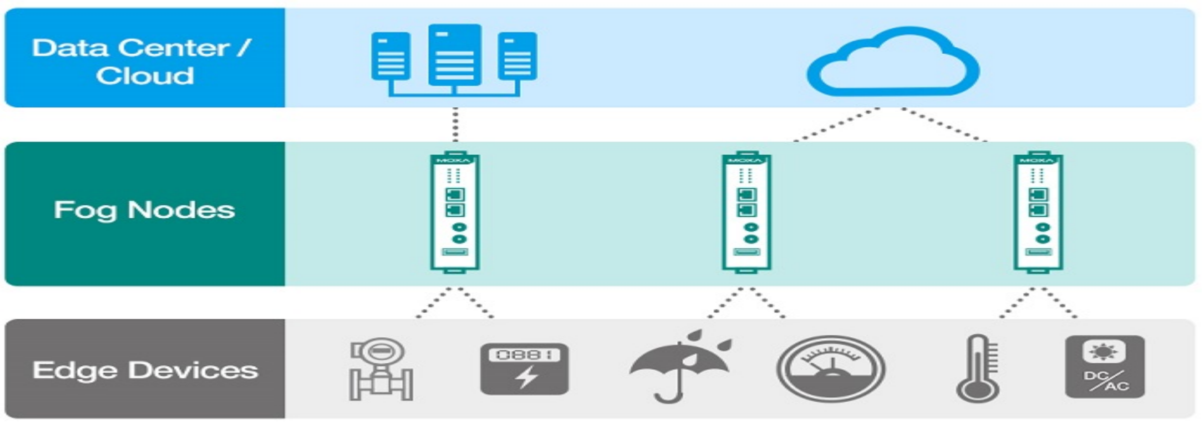

Figure 6: Fog Computing

It can happen that the edge devices are sending too much data at very high rate. Therefore the processing engine at fog level is not treating the data fast enough because the processing of a huge amount of data is taking too much time. The Complex Event Processing can be put at the

Edge devices level. The terms Fog and Edge computing have different meanings depending on the article. In some article, edge computing refers to the enabling technologies allowing computation to be performed at the edge of the network,on downstream data on behalf of cloud services and upstream data on behalf of IoT services.[**Edge Computing - Vision and Challenges**] The limit between Fog and Edge computing is not clear. In this state-of-the-art we consider the Fog to be a higher layer on the top of physical devices like a server on a private network.

4.3 Edge Computing

The Edge Computing places the computing complexity at the Physical Layer. It can be an IoT gateway where data is pre-processed before being pushed to the Cloud or whatever. For example, in e-health some sensors can be put in a human body and the informations coming from the sensors are sent to a smartphone. The smartphone is processing those information before sending them. This method reduces the network traffic to the Fog Nodes and/or the Cloud because less informations will be transferred. Therefore it reduces the risk of data congestion and the CEP engines are not flooded. Edge Computing is generally used by industries in Cyber Physical Systems[1].

5 Semantic Publish-Subscribe Architecture

The article "A Semantic Publish-Subscribe Architecture" [**A Semantic Publish-Subscribe Architecture**] suggests an interesting asynchronous architecture based on the Publish-Subscribe design pattern. The publish-subscribe pattern in software architecture is a messaging pattern where the sender of the message doesn't know the receiver. On the other side, the receiver doesn't know the sender as well. The publisher is sending a message on a message queue where subscriber(s) are listening to. In Internet of Things solutions, the producers and consumers can be the connected objects themselves but it can also be an application layer in the Cloud for example. The connected objects are publishing events on event topics and the application layer processes those events.

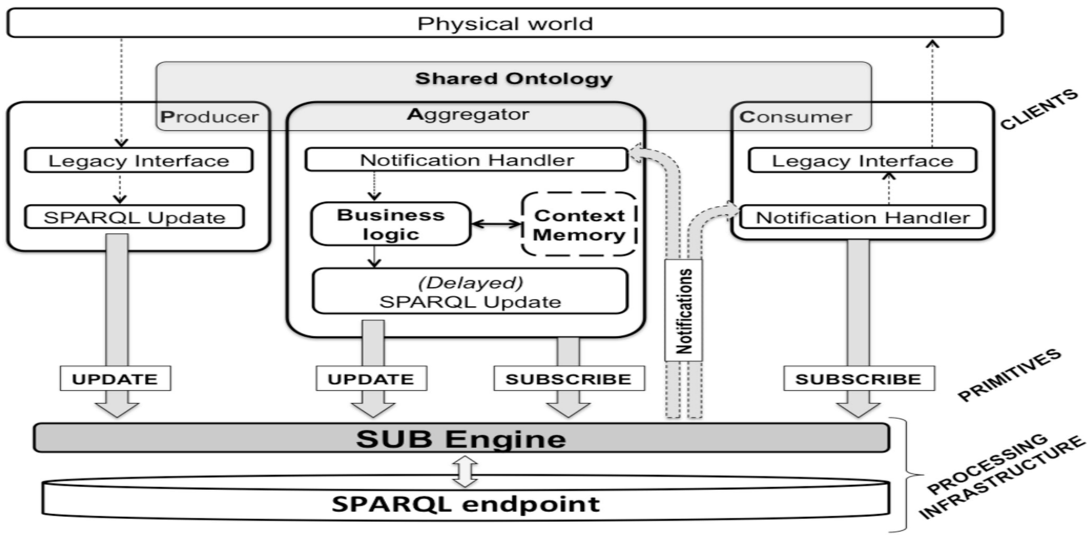

Figure 7: Semantic Publish Subscribe Architecture[**article_SPS**]

[1]Cyber-Physical Systems (CPS) are integrations of computation and physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa [4]

The SPS architecture prone the modularity, extensibility and cost-effective vision. It splits the physical world from its digital representation. The primary characteristic consists of splitting clients into 3 categories: consumers, producers and aggregators. The business logic is driven by the aggregators. The producers and the consumers are the bridge between the physical world and the virtual representation of the systems. This principle keeps the business logic at the aggregator level in order to keep the clients and producers as simple as possible. It gives to the producers the unique responsibility to send data to an upper layer without executing any business logic. Hence, this system is easier to extend because only the aggregator layer is impacted if a business feature is added. Then, the consumers and producers can be shared by different applications.

The role of the producers is to collect the data from the sensors. The producer can optionally make a local processing where it encapsulate the data to a semantic format. After that, the producer send the information to the SUB Engine that will store the data in a Triplestore. Contrarily, the consumers are listening to the concrete events coming in the SUB Engine that can be a result of a processing done by an aggregator. After receiving an notification, the consumers extract the raw data and send it to a devices through a legacy interface.

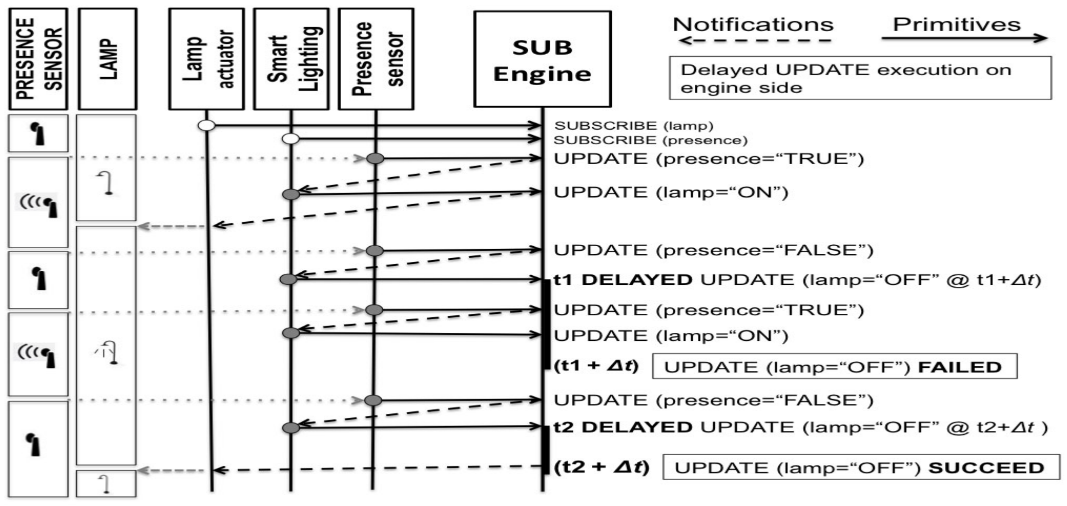
PRESENCE SENSOR
LAMP
Lamp actuator
Smart Lighting
Presence sensor
SUB Engine
Notifications
Primitives
Delayed UPDATE execution on engine side
SUBSCRIBE (lamp)
SUBSCRIBE (presence)
UPDATE (presence="TRUE")
UPDATE (lamp="ON")
UPDATE (presence="FALSE")
t1 DELAYED UPDATE (lamp="OFF" @ t1+Δt)
UPDATE (presence="TRUE")
UPDATE (lamp="ON")
(t1 + Δt) UPDATE (lamp="OFF") FAILED
UPDATE (presence="FALSE")
t2 DELAYED UPDATE (lamp="OFF" @ t2+Δt)
(t2 + Δt) UPDATE (lamp="OFF") SUCCEED

Figure 8: Sequence diagram of Smart Lightning[**article_SPS**]

The aggregators are listening to the events sent by the producer through the SUB Engine. Each aggregator subscribes to each event type they need for their business process. The fig.8 exposes a simple use case of Smart Lightning where each type of clients described previously is used. The presence sensor acts as a producer and store the presence state in the RDF Store. The Lamp actuator acts as a consumer and subscribe to the Lamp events. The Smart Lightning acts as an aggregator to handle incoming events and triggers a business process based on those events.

Scenario:
1. The Lamp Actuator is subscribing to the Lamp events and the Smart Lightning aggregator is subscribing to the Presence events. 2. The Presence sensor is Sending an Event to the SUB Engine that a presence is detected. 3. The presence event is captured by the Smart Lighting aggregator that will send an event to switch on the lamp. 4. The "Switch ON the lamp" event is captured by the Lamp actuator that communicate the information to the connected lamp. 5. The presence sensors is sending that no presence is detected. 6. The Smart Lightning schedules to send an event to switch off the light after x seconds if no presence is detected during that time laps. 7. A presence is detected, then the Smart Lightning cancels his event scheduling. 8. The presence sensors is sending that no presence is detected. 9. The Smart Lightning schedules to send an

event to switch off the light after x seconds if no presence is detected during that time laps. 10. The Smart Lightning Aggregator sends the "Switch off lamp" event. 11. The Lamp actuator is switching off the light.

6 Problematic

Prendre des exemples d'articles que j'ai analysé pour souligner des points

6.1 Scalability

One of the main challenge in IoT is the increasing of connected objects. More connected objects means that more data is sent periodically. Therefore the load of data to process increases. The Complex Event Processing is one of the answer to this problematic. The CEP will filter, aggregate, summarize the information and send it to an application layer. Then, if we have more connected objects, more complex events will be sent to this application layer. The use-case of a connected building or hospital is a great example. An initial installation can be done in a building but after a while, the connected objects increase. Thus, the application layer could be overflowed by the number of events and accordingly the processing time may take too much time. The e-health sector needs to have a fast, robust, fault tolerant and scalable infrastructure. The scalability of an IoT architecture is really important because an IoT solution is growing at the same time as the physical world. Then, the virtual world designed in an Internet of Things solution should be adapted easily with the today world and the world of tomorrow.

All the infrastructure of an IoT solution should be flexible. The today containerization technologies offer the possibilities to do so. Kubernetes and Docker Swarm offers the possibility to manage such type of infrastructure. These technologies enable the ability to scale up vertically in term of service instance. Each instance of a containerized service can be replicated to share the load of the data to process for example. If at a moment t, a lot of data is coming to a specific service, this one can be scaled up automatically to load-balance the incoming requests.

6.2 Reliability

The Internet of Things needs to scale horizontally in term of hardware resources. The aforementioned containerization technologies gives the ability to create clusters. Those clusters can be formed initially by 5 hosts for example. If the resources to process the data are not sufficient, it is possible to add more servers at runtime. On a traditional client-server architecture, the client is connecting to one and exactly one server. Nowadays, a big and complex architecture cannot rely on one unique physical (or virtual) machine. The clients (connected objects) need to have the possibility to connect on a cluster of machines containing the IoT Platform. Then, even if a host is shutting down, all the application is still running normally. There is no single point of failure.

6.3 Monitorability

An Internet of Things solution can be really complex. The complexity of such system can be explained by the presence of a huge number of connected devices. At an upper layer, the business process consuming the events generated by the devices can be really hard because a lot of type of events can be implicated for one operation. Therefore, if an error occurs somewhere in this infrastructure it reveals to be really hard to find the defects. An accurate knowledge of all processes has to be known. Then, the IoT solutions need to have a very simple and well designed monitoring tools. Those monitoring tools need to cover differents aspects. It needs to show relevant data for the end user if something wrong happened and why. A second level of monitoring has to be defined for more qualified people. Such as the error logs, events and devices metrics.

6.4 Extensibility

6.5 Device independence

Les devices doivent être agnostique l'un de l'autre (car doit être interchangeable), pas de communication one-to-one (ou alors via gateway), privilégier pub-sub, se baser sur des techno Open-Source

6.6

Parler de l'idée envoyée par mail mais de manière structurée + comment quantifier + questions et sous questions

7 Research Methodology

8 Conclusions

List of Figures

References

[1] R. Deng et al. “Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing”. In: *2015 IEEE International Conference on Communications (ICC)*. June 2015, pp. 3909–3914. DOI: 10.1109/ICC.2015.7248934.

[2] D. Giusto et al. *The Internet of Things*. Springer, 2010.

[3] An Lam and Øystein Haugen. “Complex Event Processing in ThingML”. In: vol. 9959. Oct. 2016, pp. 20–35. ISBN: 978-3-319-46612-5. DOI: 10.1007/978-3-319-46613-2_2.

[4] Edward A. Lee. “Cyber Physical Systems: Design Challenges”. In: (Jan. 2008).

[5] D. Robins. “Second International Workshop on Education Technology and Computer Science”. In: 2010.

[6] I. Schmerken. *Deciphering the myths around complex event processing*. Wall Street & Technology, May 2008.

[7] Wu Y et al. “RFID enabled traceability networks: A survey”. In: (2011).