AllJoyn Lambda: an Architecture for the Management of Smart Environments in IoT

Massimo Villari, Antonio Celesti, Maria Fazio, Antonio Puliafito
DICIEAMA, University of Messina
Contrada di Dio, S. Agata, 98166 Messina, Italy.
e-mail: {mvillari, acelesti, mfazio, apuliafito}@unime.it

***Abstract*—The increasing number of everyday embedded devices that are interconnected over the Internet leads to the need of new software solutions for managing them in an efficient, scalable, and smart way. In addition, such devices produce a huge amount of information, causing the well known Big Data problem, that need to be stored and processed. This emerging scenario, known as Internet of Things (IoT), raises two main challenges: large-scale smart environments management and Big Data storage/analytics. In this paper, we address both challenges, proposing AllJoyn Lambda, a software solution integrating AllJoyn in the Lambda architecture used for Big Data storage and analytics. In order, to describe how the architecture works, a software prototype integrating the AllJoyn system with MongoDB and Storm is presented and tested, analyzing a "smart home" case of study. Finally, we will focus on how can be possible to manage embedded devices in a smart fashion processing both batch and real-time data.**

***Keywords*-Internet of Things, Smart Environment, Big Data, AllJoyn, MongoDB, Apache Storm.**

I. INTRODUCTION

Real, digital and virtual worlds are converging to create environments that make cities, energy, transport and many other services smart. In this context, the Internet of Things (IoT) is a very challenging paradigm that enables things, i.e., embedded devices, to be ideally available anytime and anywhere to anyone. Integrating heterogeneous systems is the main goal of IoT. Indeed, it enables the development of user-driven applications connecting several sensors and objects [1]. The IoT paradigm is continuously evolving and it is leading towards the concept of Internet of Everything (IoE), where, besides the embedded devices, even people, data, and processes become things and cooperate in the ecosystem.

To support the needs of people in a smart IoT environment, embedded devices should be able to modify their behavior according to the state of the environment and people requests. AllJoyn is very engaging technology aimed at the IoT and looking at the IoE. It is an open source project that provides a framework for running distributed applications across heterogeneous embedded devices, offering interesting solutions for networking, mobility, security, and dynamic configuration issues. However, the AllJoyn technology is not enough to manage complex smart IoT environments. In fact, AllJoyn does not scale well because it does not support communications among devices belonging to different broadcast domains (i.e., belonging to different subnets) and it does not provide any feature for the storage and real-time analytics of huge amount of data (Big Data problem). Thus, when the number of devices and information acquisition frequency increase, data management becomes quite hard.

Since AllJoyn is not able to address two of the main challenges of the IoT, that are large-scale smart environment management and Big Data storage/analytics [2], this paper proposes a new scalable solution that integrate the AllJoyn system with the Lambda architecure [3], thus enabling Big Data storage, processing, and real-time analytics. More specifically, our solution includes MongoDB, that is a well known and very efficient distributed NoSQL database, and Apache Storm, that is a distributed system for real-time processing of data streams. Storm allows us to implicitly overcome the limitations of AllJoyn, giving a communication platform for devices belonging to different broadcast domains. In addition, Apache Storm support the real-time data stream processing. Finally, the system enables the heavy storage and processing of Big Data by means of MongoDB. The proposed software architecture well fits multiple IoT smart environments enabling the development of different context-aware applications and services. Furthermore, it is able to adapt its activities to the evolution of a smart environment, through a continuos monitoring and information gathering over specific the areas of interest.

The rest of the paper is organized as follows. Section II describes related works. In Section III, we motivate our work, providing and overview of AllJoyn and highlighting its main features and disadvantages for the development of smart environments in IoT. The AllJoyn Lambda architecture is presented in Section IV. A system prototype for a smart home case of study is presented in Section V. Section VI concludes the paper.

II. RELATED WORKS

The heavy penetration of sensing devices into Internet applications [4] causes the explosion of the amount of data to be stored and processed [5]. The authors of [6] provided a global model with a dynamic interoperability disregarding how the global view should be accomplished. Their decision-maker is able to process a huge amount of incoming data, but it is not clear how such capabilities are practically addressed (i.e., scalability problems). An extension of the Ubiquitous Sensor Networks framework that leverage the Sensor Web Enablement (SWE) standard along with the Session Initiation Protocol (SIP) protocol is presented in [7]. However, one of the main problems of the SIP protocol is related to network constrains.

Several solutions available in literature propose middleware for IoT purposes. In [8], the authors introduce a novel network architecture based on an M2M Gateway and discuss it in relation to smart building applications. The proposed network architecture improves services for users and offers new opportunities for both service providers and network operators. A cognitive management framework for IoT is proposed in [9], where dynamically-changing real-world objects are represented in a virtualized environment, and cognition and proximity are used to select the most relevant objects for the purpose of an application in an intelligent and autonomic fashion.

An example of Big Data processing in the Cloud is presented in [10], where the computation framework called Sailfish is proposed to improve the disk performance for large scale Map-Reduce computations. In [11], the authors face up the Big Data problem in e-health scenarios looking at non-SQL DBs as key solution toward the full development of IoT, and, specifically, they investigate on how to shift towards the Web of Things.

III. AllJoyn: Main Features and Limits for IoT

The concept of IoT was coined at the Massachussetts Istitute of Technology (MIT) in 1999, when a consortium of research, the Auto-ID Center was created with the purpose to study the RFID Technology for connecting objects, called “things”, over the Internet. Afterwords, other technologies have been considered to connect objects over the Internet, such as Near Field Communication (NFC), Wi-Fi, Bluetooth, QR code, and so on. Currently, IoT covers multiple application contexts (e.g., personal area networks, home automation, industrial production, smart cities, sensing infrastructures, etc), and it implies the need to develop new systems with distributed embedded intelligence.

A. *AllJoyn overview*

AllJoyn is one of the major promising framework for the development of IoT systems. It is an open source project for the development of a universal software framework aimed at the interoperability among heterogeneous devices, dynamic creation of proximal networks and execution of distributed applications. The framework provides a common interface towards smart devices, looking at the evolution of the IoT. The main concept of proximity and mobility identify the environment for the development of ad-hoc, proximity-based, and peer-to-peer (P2P) applications independently from the adopted communication system. Considering that it is hard to establish a P2P system over a Bluetooth networks, that an open standard for P2P does not exist, and that the P2P is not proximal, AllJoyn aims to become the standard for the communication in mobile P2P systems. AllJoyn offers the following main mechanisms:

- discovering of proximal devices and applications;
- ability to adapt the framework to specific devices;
- data transport between devices through Bluetooth, Wi-Fi, and other communication technologies;
- interoperability between different operating systems;
- efficient and secure data exchange through D-Bus.

The framework enables the development of multiple applications including media sharing, chat rooms, proximal applications, multi-player games, social applications, domotics applications, and so on. Figure 1 shows the AllJoyn architecture. The physical layer is responsible to manage IP connections

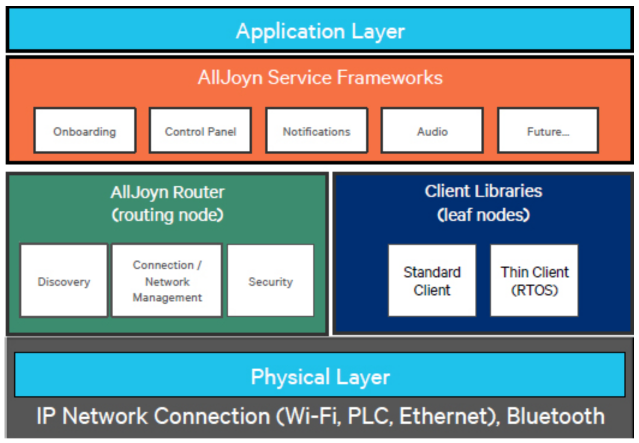
Application Layer
AllJoyn Service Frameworks
Onboarding
Control Panel
Notifications
Audio
Future...
AllJoyn Router (routing node)
Discovery
Connection / Network Management
Security
Client Libraries (leaf nodes)
Standard Client
Thin Client (RTOS)
Physical Layer
IP Network Connection (Wi-Fi, PLC, Ethernet), Bluetooth

Fig. 1. AllJoyn architecture.

using different communication technologies (e.g., Bluetooth, Wi-Fi, etc). The AllJoyn Routing component, placed on top of the physical layer, is responsible for discovery, connection management, and security. On top of the physical layer, the Standard Libraries enable the development of two types of client applications: AllJoyn Standard Client (AJST) and AllJoyn Thin Client (AJTC). The AJST applications run on general-purpose system (e.g., Microsoft Windows, Linux, OS X, Android, iOS, etc). Each AJSC application requires a Bus Daemon for internal or external communications (bundled). The AJTC connects embedded systems by means of micro-controllers installed, e.g., on household electrical appliances, televisions, thermostats, and many other devices. Since, these applications are thought for devices with limited hardware capabilities, it is not possible to use Bus Daemons running on devices. In fact, the multi-threading capabilities required by the Bus Daemon might overload the device. Thus, AJTC uses a simple Bus Attachment to use the Bus Daemon running on another device: this means that an AJTC application needs to rely on an AJSC to work. Figure 2 shows an example of AllJoyn system including two hosts, i.e., *Host A* and *Host B*, and two embedded system, i.e., *Embedded System A* and *Embedded System B*. *Embedded systems A* and *Embedded System B* join the communication bus by means of two Bus Daemons running on *Host B*.

The features offered by both AllJoyn Routing and Client Libraries are captured by the Service Framework layer to provide the following interfaces:

- Onboarding. interface that is responsible to discover AllJoyn devices, configure them, and collect data;
- Notifications: interface that sends/receives human-readable messages configuring priorities, preferences, and that supports the interaction triggered by a notification;
- Control panel: GUI that manages devices’ parameters

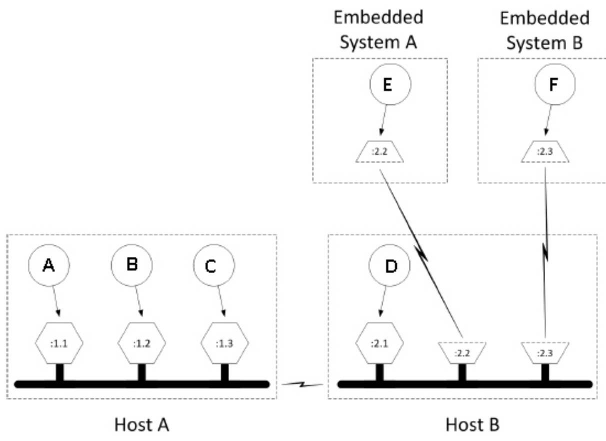

Fig. 2. Example of AllJoyn system including AJSC and AJTC applications.

- Audio: interface that is responsible to send/receive simple audio messages.

As previously introduced, AllJoyn uses D-Bus for the communication among devices. D-Bus allows us to develop Object-Oriented software independently from the used programming language. In particular, the Bus Daemon of an AJSC applications is responsible for message routing and namespace management. This enables developers to build an ad-hoc bus based on proximal discovery mechanisms. The connection of applications over the D-Bus happens according to a shared namespace. Bus Daemons send advertising messages looking for devices with particular unique well-known names included in the namespace. Once an application, i.e., AJST or AJTC, with a particular name responses to the Bus Daemon, a connection can be established. The connection takes place by means of a session mechanism that allows applications to send/receive events by means of RPC calls. Applications can be developed implementing the interfaces (including methods, signals, and properties) offered by the Service Framework layer.

B. *Limits of the Framework*

AllJoyn considered alone is not enough to manage complex large-scale smart IoT environments. In fact, the framework presents several limits in terms of scalability, efficiency, and security. Apart from security, whose implementation is explicitly demanded to application developers, AllJoyn is not able to address two of the main challenges of the IoT: large-scale Smart Environment management and Big Data storage/real-time analytics. AllJoyn does not scale well because it does not support the communication among devices belonging to different broadcast domains (i.e., belonging to different subnets). In fact, it is not possible to thing about a worldwide scenario of Cloud Computing including billions of IoT devices connected on the same broadcast domain. In addition, AllJoyn does not provide any data management system. The billions of devices that will be connected over the Internet by 2020 will introduce on the network a huge amount of data. Thus, it is indispensable to pick out specific data management systems able to process Big Data. To this end, system able to massively process offline Big Data and to process real-time streams of Big Data are strongly required. Both offline and real-time Big Data processing mechanisms are required to develop large-scale smart environments capable to adapt and configure themselves according to events, preferences, and actions performed by users and other "things".

IV. STORM/MONGODB/ALLJOYN: AN ARCHITECTURE FOR SMART ENVIRONMENTS IN IOT

As previously discussed, AllJoyn does not meet all the requirement of smart environments in IoT. In this Section, we describe a software architecture able to overcome such limitations considering the Lambda architecture. With the term Big Data analytics, we refer a set of processes to collect, organize, and analyse a huge amount of heterogeneous data. In a scenario that rapidly grows such as the IoT, the analysis of Big Data becomes fundamental for the development of smart systems. Currently, different architectures are available in the Big Data panorama offering particular features. Big Data analytics scenarios can be multiple, in fact, when software architects design a system, they have to choose the architecture that best fits the requirement of the scenario they want to address. MapReduce, designed by Google, is one of the most famous architecture. It adopts a master/slaves approach enabling parallel processing: a master splits a job in tasks and distribute them over different workers (the Map phase). Once the workers finished, the master collects results and produces the output (the Reduce Phase). Another solution is the Three-Level architecture. It includes three levels: online processing, nearline processing, and offline processing. The first level is responsible for the processing of real-time data streams. Typically, this level runs lightweight algorithms. The second level is responsible to process both real-time and offline data combining different database solutions (e.g., NoSQL and SQL-like). The third level is responsible for the processing of massive offline heterogeneous raw data. This level typically includes a distributed parallel processing system (e.g., MapReduce). Machine Learning is an important concept used in this architecture. In fact, the architecture adopts algorithms able to adapt themselves according to the processed data. These algorithms are often scheduled by Model Training activities. Another solution for Big Data analytics is the Lambda architecture. In our opinion, this architecture well fits the requirements of smart environments in IoT and will be the core of the software solution proposed in this paper.

A. *The Lambda Architecture*

The Lambda architecture enables to process and to storage huge amount of data. This distributed and scalable architecture is based on three principles:

- Fault-tolerance. The system does not have to lose data due to human errors and hardware failures.
- Data immutability. The system has to permanently store data that are immutable. Update and delete tasks on data are not allowed.
- Recomputation. Due to fault-tolerance and immutability, for each query, data have to be recomputed. Thus, we can consider *query=function(all data)*.

Figure 3 shows the Lambda architecture. Differently from the

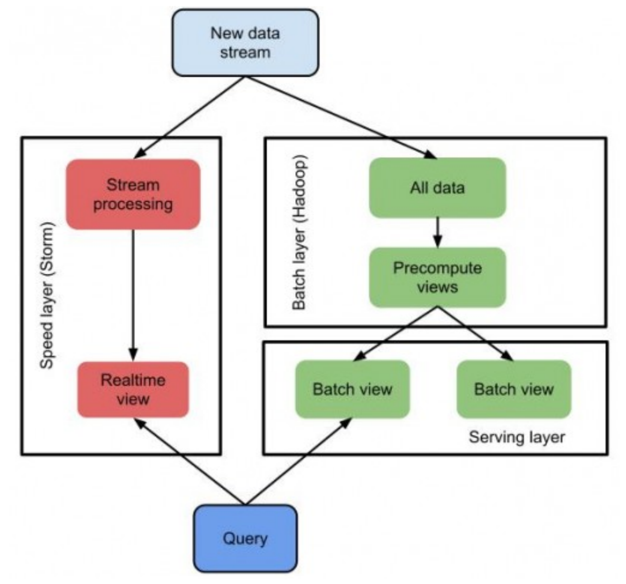

Fig. 3. Lambda Architecture.

Three-Level architecture, Lambda includes only two levels of latency, i.e., Speed (analogous to Online) and Batch (analogous to Offline). In fact, a layer analogous to the Nearline is not present. The Batch layer is responsible to store the master dataset, that is immutable and in constant growth, and to produce data views. When new data arrive, new views are elaborated processing the whole master dataset. Instead, the Speed layer is responsible to produce data views of the most recent real-time data. This layer uses an incremental model. The produced real-time views are temporary, in fact, once data are forwarded to the Batch layer they are deleted. The serving layer is responsible to add indexing metadata to batch views. Apache Storm

B. Integrating AllJoyn in Lambda

In order to develop an architecture for smart environments in IoT, according to the Lambda architectural model, we integrated the AllJoyn system with Storm for the processing of real-time data and MongoDB for the massively storage and processing of batch data. The system includes at low level several AJTC applications for controlling different devices, e.g., fans, windows, blinds, thermostats, lamps, and so on. All the AJTC applications belonging to a particular environment are connected to the Bus Daemon running on an environmental AJSC application responsible to manage that particular environment, e.g., a room, a shop, an office, and so on. Furthermore, the environmental AJSC applications can be connected to a centralized AJSC application (e.g., a home, a moll, etc) that collect sensing data in MongoDB. AJSC application can be deployed in a distributed fashion. In addition, both the centralized AJSC applications and the various environmental AJSC applications are connected to the Storm system for a global management. Thus, our system allows us to consider three levels of management: environmental, centralized, and global.

Thanks to Storm and MongoDB, we can manage scalable environments. The Storm cluster includes two types of node: Master and Worker. The Master node runs the Nimbus daemon that is responsible for distributing tasks to the Worker nodes that run a Supervisor daemons. The Coordination between the Master node and Slave nodes is performed by several distributed Zookeeper nodes. This distributed architecture makes the system highly scalable. Figure 4 shows an example of Storm cluster. Storm processes data using the concept of

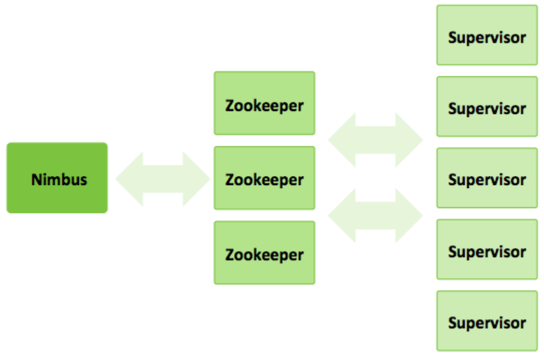

Fig. 4. Example of Strom cluster.

Tuple, i.e., an ordered list of elements. A sequence of Tuples define a Stream. The system provides functionalities to turn a Stream into another Stream by means of Spout and Bolt nodes. Logically, a network of Spouts and Bolts by means of Stream grouping techniques, is defined Topology. In particular, the Spout is a type of node acting as a source of Streams that have to be processed in the Topology. The Bolt is a type of node responsible for the computation that happens in the Topology. It receives data from Spouts and can forward the result to other Bolts for further computations. To collect real-time data coming from AJTC applications, we integrated in each Spout node an AJSC. Thus, for each environmental AJSC, we integrated in the Storm system a particular Topology. Figure 5 shows an example of logical Storm architecture where Spout nodes collects data coming from different AJSC applications. Considering a scalable scenario, we can have two types of

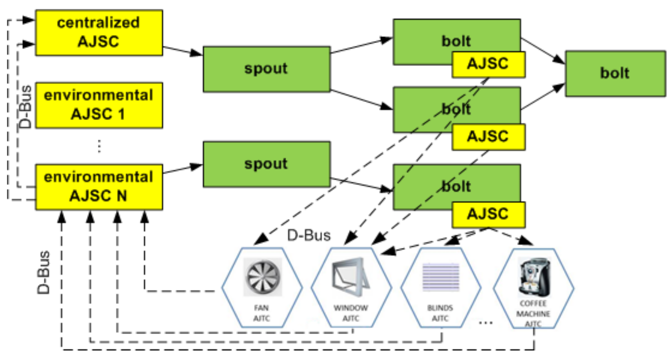

Fig. 5. Example of logical Storm architecture where Spout nodes collects data coming from different AJSC applications.

super-peer overlay networks. The first one includes different environmental AJSC applications connected to the Storm system, whereas the second one includes different centralized

AJSC connected to the Storm system.

As previously described, each centralized AJSC application is responsible to control several environmental AJSC applications, each one controlling in turn several AJTC applications running on several embedded devices. In addition the centralized AJSC application is responsible to store the collected sensed data in MongoDB for batch data processing. Thus MongoDB will store data in a collection called “device groups”. In addition, MongoDB stores data regarding how to control devices in a smart fashion. In fact, data patterns that define the behaviour of devices are stored by means of three collections: “regular patterns”, “event patterns”, and “automated patterns”. These collections will be accessed by the Strom system during the real-time data processing of data coming from embedded devices ,i.e., AJTC applications. If a sensing device matches a data pattern, a set of actions will be performed by the Storm system. To this end, in each Bolt node, an AJSC will send actions to devices involved in the data pattern. In the following, we describe the three types of data patterns that the architecture supports:

- **Regular patterns** includes a set of actions performed by several embedded devices running AJTC applications that have to be periodically repeated. E.g., from 14th July, 2014 to 18th July, 2014 the coffee machine has to be switched on at 6:45;
- **Event-based patterns** includes a set of actions performed by several devices according to particular events. For example, if sensors detect smoke all windows must be opened;
- **Automated patterns** includes actions not configurable by users, that are triggered according to complex algorithms. E.g., the curtains in a flat can be automatically moved according to the intensity of the external light.

In addition, it is possible to apply a priority to each data pattern.

V. A Smart Home Case of Study

In this Section, we describe the major implementation highlights for the development of an AllJoyn/Storm/MongoDB prototype. In particular, we will describe how to arrange the basic system and how to develop data patterns for the smart real-time management of embedded devices. In particular, we will consider a smart home case of study. Let us consider a residence including different smart flats. We want to control and monitor the behaviour of devices according to rooms, flats, and the whole residence. The first proximal network that we consider is the room. To this end, we assume that in each room a tablet running Android manages different devices. Android runs an AJSC application and a Bus Daemon used to connect the AJTC applications running on embedded devices. Figure 6 shows an example of an AJTC running on a thermostat that establish a connection with the Bus Daemon running in the Android AJSC application. Figure 7 shows an example of the of environmental AJSC application of a kitchen running in an Android tablet. The android application allows us to control, in first instance, the electrical applications connected to the proximal network of the kitchen. For each smart flat, a

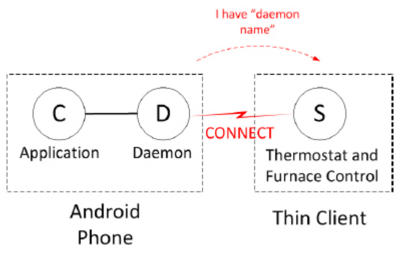

Fig. 6. Example AJTC application running in a thermostat that establish a connection with a Bus Daemon running in an Android application.

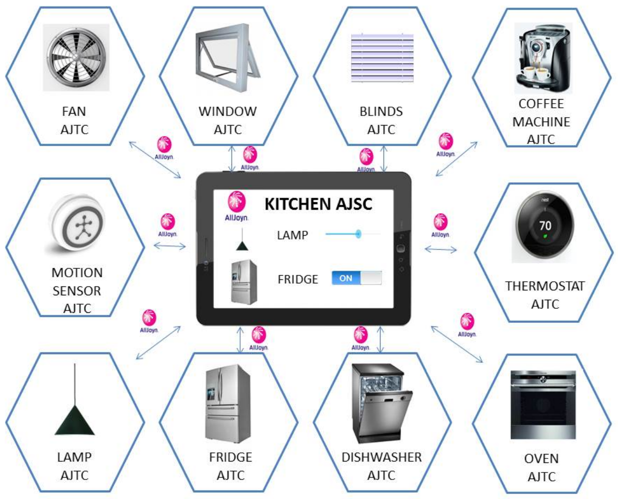

Fig. 7. Example of AJSC application controlling the electrical appliances of a kitchen.

centralize AJSC application allows us to control and manage the AJSC application running on the tablets of the various rooms. It is responsible to facilitate the monitoring the all rooms and to apply data patterns in real-time. In addition, it is also responsible to collect sensed data and to store them in MongoDB. The whole smart flat is managed by a “Home Cluster” that consists in a distributed Storm/MongoDB system. Figure 8 shows a Home Cluster including three hosts. In each

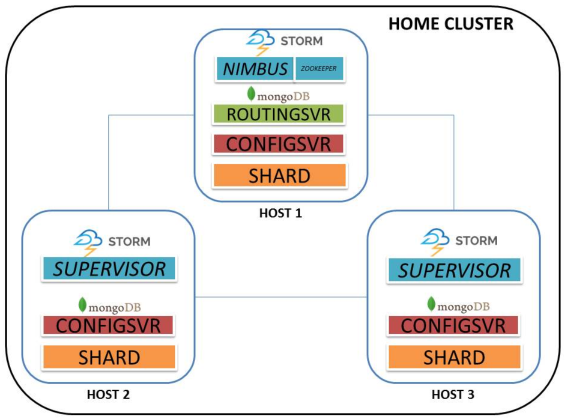

Fig. 8. Example of Home Cluster including three hosts.

host, we configured pieces of Storm and MongoDB software.

In particular, we configured MongoDB to work with three shards. i.e., replica sets. In addition, we configured one host acting as Storm Master, i.e., running the Nimbus daemon, and two hosts acting as Storm Workers, i.e., running Supervisor daemons. In each, Storm host, we integrated particular AJSC in both Spout and Bolt nodes in order to collect data coming from devices and to control them. The whole Smart Home system is shown in Figure 9. Thus, it is possible to control

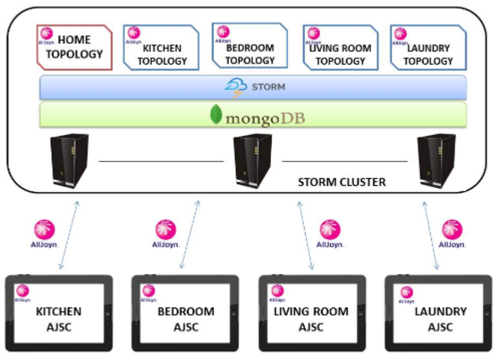

Fig. 9. Example of Smart Home system.

single environments in real-time and to analyse the behaviour of the devices considering particular time intervals. To simply the management of each smart environment, we developed a Storm topology for each room. In addition, we develp a web interface to control the whole system, i.e., for setting up data patterns and for monitoring devices.

In order to test the system, we emulated the behaviour of different electrical appliances placed in a kitchen and we monitored them using the web interface. The graph on Figure 10 shows the programmed behaviour of devices in the kitchen considering 24 hours applying a regular pattern. Through the environmental AJSC of the kitchen, data coming from devices are sent to a Storm Spout that forwards them to a Storm Bolt that, by means of an AJSC switches on of the dishwasher and the coffee machine at particular time intervals according to the data pattern retrieved in MongoDB. In addition, thanks to

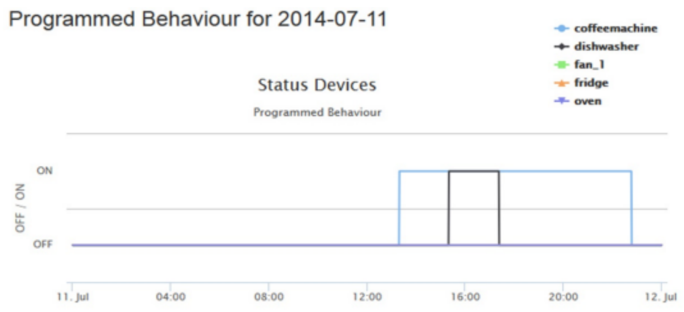

Fig. 10. Example of regular pattern monitoring.

MongoDB, it is possible to analyse batch data. For example, the graph of Figure 11 shows the energy consumption of all the electrical appliances of the kitchen on July 10th, 2014. Similarly, the AllJoyn Lambda system can be extended for the smart management to the whole residence.

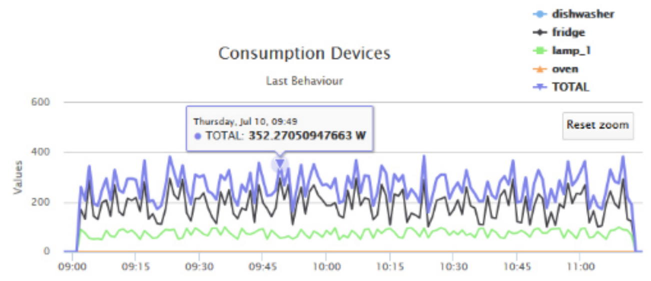

Fig. 11. Example of energy consumption monitoring.

VI. Conclusion

In this paper, we discussed the limitation of AllJoyn considering the development of large-scale smart environment in IoT. In particular, we proposed a solution inspired to the Lambda architecture, integrating AllJoyn with MongoDB and Storm. In addition, we discussed how the system allows us to overcome the limitation of AllJoyn in terms of large-scale Smart Environment management and Big Data storage/analytics considering both real-time and batch information. Finally, a smart home case of study was analysed.

References

[1] D. Mulfari, A. Celesti, M. Fazio, and M. Villari, "Human-computer Interface Based On Iot Embedded Systems For Users With Disabilities," in *PROCEEDINGS of International Conference on IoT as a Service (IoTaaS), Springer*, October 2014.

[2] M. Fazio, A. Celesti, M. Villari, and A. Puliafito, "The need of a hybrid storage approach for iot in paas cloud federation," in *28th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pp. 779–784, May 2014.

[3] Lambda Architecture, http://lambda-architecture.net/.

[4] R. Tabish, A. Ghaleb, R. Hussein, F. Touati, A. Ben Mnaouer, L. Khriji, and M. Rasid, "A 3g/wifi-enabled 6lowpan-based u-healthcare system for ubiquitous real-time monitoring and data logging," in *Biomedical Engineering (MECBME), 2014 Middle East Conference on*, pp. 277–280, Feb 2014.

[5] A. L. Tu'n, H. Quoc, M. Serrano, M. Hauswirth, J. Soldatos, T. Papaioannou, and K. Aberer, "Global sensor modeling and constrained application methods enabling cloud-based open space smart services," in *Ubiquitous Intelligence Computing and 9th International Conference on Autonomic Trusted Computing (UIC/ATC), 2012 9th International Conference on*, pp. 196–203, Sept 2012.

[6] D. Ballari, M. Wachowicz, and M. A. Manso, "Metadata behind the interoperability of wireless sensor network," *Sensors*, vol. 9, pp. 3635–3651, 2009.

[7] J. M. Hernández-Muñoz, J. B. Vercher, L. Muñoz, J. A. Galache, and Presser, "Ubiquitous sensor networks in ims: an ambient intelligence telco platform.," in *ICT Mobile Summit*, (Stockholm), 10-12 June 2008.

[8] R. Fantacci, T. Pecorella, R. Viti, and C. Carlini, "A network architecture solution for efficient iot wsn backhauling: challenges and opportunities," *Wireless Communications, IEEE*, vol. 21, pp. 113–119, August 2014.

[9] P. Vlacheas, R. Giaffreda, V. Stavroulaki, D. Kelaidonis, V. Foteinos, G. Poulios, P. Demestichas, A. Somov, A. Biswas, and K. Moessner, "Enabling smart cities through a cognitive management framework for the internet of things," *Communications Magazine, IEEE*, vol. 51, pp. 102–111, June 2013.

[10] S. Rao, R. Ramakrishnan, A. Silberstein, M. Ovsiannikov, and D. Reeves, "Sailfish: A framework for large scale data processing," in *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, (New York, NY, USA), pp. 4:1–4:14, ACM, 2012.

[11] M. Diaz, G. Juan, O. Lucas, and A. Ryuga, "Big data on the internet of things: An example for the e-health," in *Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS 2012)*, pp. 898–900, July 2012.