A Novel IoT Architecture with Pattern Recognition Mechanism and Big Data

Alberto M. C. Souza¹ and José R. A. Amazonas²

¹*Escola Politécnica, University of São Paulo - USP and Cruzeiro do Sul University - Brazil*

²*Escola Politécnica, University of São Paulo - USP, Brazil*
Corresponding Authors: {linuxstring; joserobertoamazonas}@gmail.com

Received 28 April 2015; Accepted 12 May 2015;
Publication 29 May 2015

Abstract

One of the most important IoT challenges is scalability. This paper addresses this issue by introducing pattern recognition services into the lower layers of the IoT reference model stack and reduces the processing at the higher layers. The work adopts the reference model developed by the IoT-A project and the LinkSmart Middleware platform. The new architecture implementation extends the LinkSmart introducing a pattern recognition manager that includes algorithms to estimate parameters, detect outliers, and to perform clustering of raw data from IoT resources. The new module is integrated with the Big Data Hadoop platform and uses Mahout algorithms implementation.

Keywords: Internet of Things, Big Data, Architecture.

1 Introduction

The Internet of Things (IoT) is a new communication paradigm in which the Internet is extended from the virtual world to interface and interact with objects of the physical world. A huge amount of applications and services can then be developed and simultaneously an immense set of challenges must be overcome to make the IoT come true. IoT involves different areas of knowledge as pervasive computing, network communication, object identification, data processing, among others.

Journal of Machine to Machine Communications, Vol. 1, 245–272.

doi: 10.13052/jmmc2246-137X.134

© 2015 River Publishers. All rights reserved.

Pervasive (or ubiquitous) computing has much in common with the computing invisibility proposed by Weiser in [29], where embedded computers get the information from the environment and use them dynamically as a computing model. In this way, the computers can perform smart actions, making the environment more flexible and able to adapt to the current context [13].

Gluhak, Bauer, Montagut, Stirbu, Johansson, Vercher and Presser in [6], show that communication method with the physical world will involve large sets of devices and the trend is that this number will increase dramatically in the upcoming years [4].

Raw data generated by different devices must be processed or stored to further later processing. According to Botts, Percivall, Reed and Davidson, in [2], raw data's metadata are needed to enable future analysis, as for example, metadata with geographic location and time reference. In [6] the author states that new efficient mechanisms and policies should be included in the network infrastructure to manage and store these data.

According to Smith [21] data management in IoT is crucial aspect. Considering a world of interconnected objects which constantly changes many kinds of information, the volume of generated data and involved processes, the management of data becomes critical. New services to process and analyze the massive data generated by the communication between devices will be needed. These services will have open interfaces and will have to be able to provide a simple integration between many applications.

In this context we introduce pattern recognition mechanisms in the IoT architecture. In this paper implement pattern recognition algorithms to estimate values, detect outliers and to perform clustering [22]. The chosen algorithms focus on highlighting relevant information to analyze and predict behavior of the human individuals, social communities, animals, computer networks, traffic, consumption, to implement security mechanisms, and to provide assistance or automation, among others applications as those mentioned by Roussos in [19].

The pattern recognition mechanisms are implemented in the lower layers of the IoT model, namely the physical, middleware and services layers [22] and use Big Data technology for distributing processes. The focus of the paper is on the architectural aspects of introducing the pattern recognition algorithms and the modular implementations enables an easy introduction of other algorithms according to needs of new applications and services. The proposed architecture based on IoT-A reference model, has become the European Commission's flagship project in the European Union's Seventh Framework Program for

Research and Development with respect to establishing an architecture for the Internet of Things [27].

The paper is organized as follows: after this brief Introduction, in Section 2, we describe the main IoT concepts, introduce the IoT-A reference model, IoT middlewares and Big Data technology. The proposed architecture and its implementation details are shown in Section 3. Conclusions and future works are presented in Section 4.

2 Background

This section introduces the main concepts related to the IoT, an IoT reference model, the description of the LinkSmart IoT middleware, and Big Data related processing techniques.

2.1 Internet of Things Concepts

As stated in [3], Internet of Things is a global network infrastructure, linking physical and virtual objects through the exploitation of data capture and communication capabilities. This infrastructure includes existing and evolving Internet and network developments. It will offer specific object-identification, sensor and connection capability as the basis for the development of independent federated services and applications. These will be characterised by a high degree of autonomous data capture, event transfer, network connectivity and interoperability, actuation and control.

Figure 1 illustrates the CASAGRAS Inclusive Model [8] reproduced from [1] and proposed in the CASAGRAS project (<http://www.iot-casagras.org>).

According to the Figure 1 and to the CASAGRAS Inclusive Model described, a real-world object has its identification ID and associated information stored, for example, in a RFID tag. It is important to realize that the identification technology is not restricted to RFID. Biometry and bar codes are other examples of ID technology that can be employed. The information is retrieved from the object by means of an interrogator that acts as a gateway device and stores the information in a host management system. The Internet is used both to allow access to the retrieved information and to search for further information and associated applications and services. The end result is that an action will take place either displaying new information and/or acting upon the object and/or the environment [1]. The whole process is context-aware and the final action depends on the object itself and its present status in the current environment.

Figure 1 CASAGRAS Inclusive Model [1].

2.2 The IoT-A Reference Model

A reference model for IoT should provide a high abstraction level to define a reference architecture. The reference model provides an understanding about the IoT domain. The model has a high level description, an associated information model which describes the information flow, and a communication model to describe the interaction between devices.

The *IoT-A* reference model consists of a set of sub-models that represent all aspects of IoT. Figure 2 which shows the interactions between sub-models [10].

According to Figure 2, the base of the IoT-A reference model is the domain model, which introduces devices, services, and virtual entities, and the relationship between them. This abstraction level is independent of technology and time. Based on the domain model, the information model defines the structure of relationships, attributes, data and data flows, and how this information is processed by the IoT system, at a conceptual level. The relevant information related to domain model's entities are defined. The information about devices, services and virtual entities are explicitly collected, stored and processed in IoT system [10].

Figure 2 Interactions between IoT-A sub-models [10].

The functional model specifies functionality groups, that are related to the domain model's entities. In addition, the possible interactions between functionalities groups are also defined as shown in Figure 3 [10].

Figure 3 shows the communication model which defines the main communication features to connect the entities defined in the domain model.

Figure 3 Possible interactions between functionalities groups [10].

This model provides a communication reference for the main interactions among the entities of the domain model. The management model provides orchestration among different functionalities groups. Finally, the security and privacy models describe the high level and abstract concepts related to trust, security and privacy in the context of IoT [10].

For the purposes of this paper, the functional model is the most important to understand the proposed new architecture.

2.3 Internet of Things Middlewares

As shown in [11], there are many middleware softwares, which are defined as software systems that provide an abstraction layer between the operating system and development applications environments in the context of pervasive computing, whose focus is to provide a usual and abstracted suite of procedures that can deal with the heterogeneity of devices and contexts of information.

We want to highlight the Internet of Things middleware *Network Embedded System for Heterogeneous Physical Devices Middleware in the Distributed Architecture - (HYDRA)* created by the FP6 IST [20], which started in July 2007 and finished in December 2010. It will be described further in Section 2.3.1.

2.3.1 Hydra project

According to [17], the first objective of the Hydra Project was the development of a software middleware based on the Service-Oriented Architecture (SOA), in which the communication occurs transparently between the lower layers.

The framework supports centralised and distributed architectures, security and trust, and model the applications development. One of the framework's development premises was its applicability in current networks and novel networks models with interconnected devices that operate with reduced computational power, energy and memory capacity.

The resulting product of this project was called LinkSmart middleware, a name that will be used to refer to the developed middleware from this point onward. Figure 4 illustrates the layered structure of the LinkSmart middleware.

According to Figure 4, the elements of the LinkSmart middleware are placed between the application and physical layers shown in the diagram. The physical layer is related to network communication resources, while the application layer contains modules related to the management of information flow, user interface, application logic and configuration details. Between the two layers is the LinkSmart middleware, consisting of three sub-layers,

Figure 4 Layered structure of the LinkSmart middleware [20].

network, service and semantics, each of them responsible for specific functions and purposes [9, 20].

Application elements describe components deployed on hardware which are performance-wise capable of running the application that the solution provider creates. This means these components are meant to run on powerful machines.

Devices elements describe components deployed inside the LinkSmart Framework. These components could be deployed in small devices which have limited resources in terms of processing power or battery life. These components have a limited set of functionalities but could also be deployed on another machine acting as a proxy for e.g. a mote where it would be highly unlikely that those managers would ever be deployed on such a resource-limited device [20].

It is worth noting that the structure of the LinkSmart Middleware is closely related to the functional model defined in the IoT-A reference model and architecture, and illustrated in the Figure 3, consisting of services and security layers, entities and management services.

2.4 Big Data

Sun and Heller in [23] define Big Data as large datasets which are hard to store, search, view and analyze, such as, for example, an air company case that collects 10 terabytes of data from sensors in a 30 minutes flight. Tracey and Sreenan show in [28] that Big Data techniques are used commercially to analyze large datasets to make decisions based on behavior analysis.

According to Smith, in [21], Big Data technology refers to processing and analysis of large datasets, which would not be analyzed or processed with conventional data analysis tools. The Big Data technology requires large computing power for processing large datasets in short or acceptable time, it involves massive parallel processing databases (MPP), data mining grids, distributed filesystems, cloud computing, Internet communication and scalable datasets.

3 A Novel IoT Architecture

In this section we introduce a novel IoT architecture that incorporates mechanisms for pattern recognition in the reference model's service layer.

The new architecture is implemented in the LinkSmart middleware which is extended by the inclusion of new pattern recognition services that implement and abstract algorithms to perform outlier detection, values estimation and clustering. This solution can apply these algorithms to data coming from any kind of environments and devices. Applications will retrieve contextualized information from the middleware rather than raw data directly from devices or the former middleware layer.

Figure 5 shows the proposed architecture implemented in the layer structure of the LinkSmart middleware.

In Figure 5 we see a new box called *Pattern Layer*, highlighted by a red rectangle. This new layer has three managers: classification, recognition and estimation, which implement the pattern recognition functionalities. At the current stage of this research the implementation focused on the application elements seen at left side of Figure 5.

Figure 5 A new layer structure of the LinkSmart middleware [20] incorporating pattern recognition mechanisms.

The algorithms to estimate values, classify and recognize behaviors, and to detect outliers [5, 26] contribute to network traffic reduction in the IoT context as the upper application layers will not receive raw data anymore pre-processed information by the LinkSmart middleware pattern services.

These algorithms have been implemented as a distributed architecture to process data using the Big Data technology. The following techniques have been used:

- linear regression to estimate values;
- k-means algorithm to cluster and contextualize values retrieved from sensors and others devices;
- clustering distance to detect outliers [5, 12, 16].

Figure 6 shows the implementation of the proposed architecture:

The most important aspect of this implementation is a new pattern recognition module inserted in the LinkSmart Middleware. This implementation

Figure 6 Implementation of the proposed architecture.

follows the architecture proposed by the IoT-A reference model, described in Section 2.2, and has the following layers:

- **Physical layer:** it is designated as resource layer and hosts sensors and smart objects. The resource manager box represents the driver or software responsible to connect with the LinkSmart and send either raw data or pre-processed data, in this case by the estimation and/or outlier detect algorithms. If data generated by the physical resource is processed in this layer, the resource manager informs the configuration manager in the LinkSmart or directly the application by means of the Control Information channel. The configuration manager or the application can change parameters in the resource manager to disable the pre-processor and the resource manager proceeds to send the raw data;
- **Middleware layer:** it is represented by the new LinkSmart middleware, which includes a pattern manager and pattern configuration manager proposed in this implementation, the event manager, which has an important role in this new architecture, and other services which have not been modified and used in this work. The pattern manager provides three

services: estimation, clustering and outlier detection. The configuration manager enables applications and the resource manager to configure parameters in the pattern manager, as the activation or deactivation of the algorithms, how much data must be stored, enable or disable the pattern recognition services, clean the stored data. After processing data, the pattern services send the contextualized information to the event manager that it will send the new event to the clients. The event manager is responsible to send the processed information or the raw data to client applications. It addresses the scalability issue because it is able to communicate to any number of client applications and inform the events of interest. The event manager can be directly accessed by the resource manager, that in this case doesn't use the pattern services. This mode of operation can be chosen by the application designer, and can be changed in the configuration manager whenever necessary.

- Application layer: it is represented by client applications and their associate configuration actions. The client applications receive the events from event manager, either raw data or processed information. The configuration actions are responsible to configure parameters in the configuration manager or to send parameters to the resource manager, to control the behaviour of the resource manager or the pattern services. This communication is bidirectional.

Details about the implementation are described further in Section 3.1.

3.1 The Pattern Manager Implementation and Big Data Processing

The LinkSmart middleware was implemented using the *OSGI* architecture [7]. To create a new module in this middleware we also use the *OSGI* architecture, extending its packages and creating *Interfaces* to build the new services.

Figure 7 shows the main class diagram of the pattern manager. This class diagram describes the LinkSmart 's packages and the new created package and designated called by *pattern*. This new package has the following classes:

- *ClassificationManager*: this is an interface, which defines the existing subscription methods in the new services. These are services implemented in the pattern manager as: register and unregister a pattern hardware identification (PHID), list the PHIDs, submit new raw data, remove data from PHID, run the algorithms, set attributes to algorithms and others;

Figure 7 Class diagram of the pattern manager including packages, interface, implementation and parameter class.

- *ClassificationManagerImpl*: this is a concrete class which implements the methods of the *ClassificationManager* interface and aggregates methods to implement an *OSGI* service. This class manages the instances with implementations of the pattern recognition algorithms.

Others classes have been developed to implement the pattern recognition algorithms and integrate them with the Hadoop platform [30]. Figure 8 shows the class diagram with the classes that implement the clustering service in the pattern module.

In Figure 8 we see the already existent *eu.linksmart.pattern* package and the new packages *clusterer* and *hadoop*. The implemented classes are:

Figure 8 Class diagram of the clustering service implementation in the pattern module.

- *PatternSubscription*: this class creates the structure with attributes to new classes which implement services in the pattern module. This class defines attributes which inform the type of algorithm (clustering, outlier or estimator), type of attributes to process (numeric, text, class or date) and *PHID* that is a unique Pattern Hardware Identification assigned to each resource;
- *PatternSubscriptionClustering*: this class extends the *PatternSubscription* class and aggregates new methods specifically to the clustering service, as the *inputInstance* to insert an instance to process, *findCluster* to find a cluster of an instance, *setAttributesClustering* to define parameters of the clustering algorithm, *runPattern* to start the clustering process, *finishedRunSubscriptionClustering* to notify if the

clustering process has finished, and *ResultPatternClassificationClustering* to return the result of the process. This class does not implement the clustering algorithm itself but incorporates a *PatternClustering* object to which it delegates the processing. This solution decouples the module from the concrete implementation and enables the easy introduction of other implementations.

- *PatternClustering*: this interface defines the structure so that new classes can implement the necessary methods to be a *PatternClustering* object. This class defines the following methods: *inputInstance*, *findCluster*, *runPattern*, *returnResultPatternClassificationClustering*, *setAttributesClustering* and *finishedRunSubscriptionClustering*;
- *ResultPatternClassificationClustering*: this class contains the attributes with all the results of the clustering algorithms. A *ResultPatternClassificationClustering* object can be sent to an application where the information about clusters may be used, to contextualize the information and create knowledge about the data processed in the middleware layer.
- *PatternClusteringHadoopImpl*: this is a very important class in the clustering service, as it implements the *PatternClustering* interface and all methods proposed for the interface. It implements the *k-means clustering algorithm* integrated with the *Big Data* processing instance of *Hadoop*. We use the *Mahout* framework implementation of the *k-means* algorithm [15]. The main method implemented in this class creates a thread to start the process in the *Hadoop* instance. When the process finishes the implemented method toggles a *flag* to inform and send the result to the *LinkSmart Event Manager*.
- *SimpleKMeansClusteringHadoop*: this class implements the integration and runs the *Kmeans* thread in the *Hadoop* instance.

This implementation architecture allows any other class that implements the *PatternClustering* interface to be plugged in the pattern module. In a previous implementation we used the *weka* classes [31], but we didn't get a satisfactory performance and we decided to use the *Big Data* technology with *Hadoop* and the *Mahout* implementations. This *Big Data* implementation allows to create a *Hadoop* cluster to increase the computational power.

The estimator service has the same structure as the one explained for the clustering service. The class *PatternSubscription* is important to couple the implementation of the algorithm to the main structure of the pattern module.

Figure 9 shows the class diagram that implements the estimator service in the pattern module.

Figure 9 Class diagram of the estimator service implementation in the pattern module.

In Figure 9 we see the already existent *eu.linksmart.pattern* package and the new packages *estimator* and *weka*. The implemented classes are:

- *PatternSubscription*: this is the same class used in the clustering service and has the same structure;
- *PatternSubscriptionEstimation*: this class extends the *PatternSubscription* class, and aggregates new methods specifically to the estimator service, as the *inputInstance* which has the same function as described in the clustering service, *estimateSubscription* to estimate a specific value of an instance, *setAttributesEstimation* to define parameters to be used by the estimation algorithm as the target attribute, *runPattern* to start the estimation process, *finishedRunSubscriptionEstimation* to notify when the estimation process has finished and *returnResultPatternClassification* to return the result of the process. This class does not implement the estimator algorithm itself but incorporates a *PatternEstimation* object to

which it delegates the processing. This solution decouples the module from the concrete implementation and enables the easy introduction of other implementations.

- *PatternEstimation*: this interface defines the structure so that new classes can implement the necessary methods to be a *PatternEstimation* object. This class defines the following methods: *inputInstance*, *estimateSubscription*, *runPattern*, *returnResultPatternClassification*, *setAttributeEstimation* and *finishedRunSubscriptionEstimation*.
- *ResultPatternClassificationEstimation*: this class contains the attributes with all the results of the estimation algorithms. A *ResultPatternClassificationEstimation* object can be sent to an application where the information about estimation may be used as a linear regression function [5].
- *PatternEstimationWekaImpl*: this is a very important class in the estimator service, as it implements the *PatternEstimation* interface and all methods proposed for the interface. It implements the *linear regression algorithm* using the *weka* implementation to process the data. The main method implemented in this class creates a thread to start the process with *weka* classes. When the process finishes the implemented method toggles a *flag* to inform and send the result to the *LinkSmart Event Manager*. It is important to point out that for this service it wasn't necessary to use the Hadoop integration, but the object-oriented structured programming allows it.

The outlier detection service has the same structure as the clustering and estimator services. Figure 10 shows the class diagram that implements the outlier detection service in the pattern module.

In Figure 10 we see the already existent *eu.linksmart.pattern* package and the new packages *outlier* and *hadoop*. The implemented classes are:

- *PatternSubscription*: this is the same class used in the clusterer and estimator services and has the same structure;
- *PatternSubscriptionOutLier*: this class extends the *PatternSubscription* class, and aggregates new methods specifically to the outlier service, as the *inputInstance* which has the same function as described in the clustering service, *isOutLier* to estimate if a specific instance is an outlier, *setAttributesOutLier* to define parameters to be used by the outlier detection algorithm, *runPattern* to start the outlier detection process, *finishedRunIsOutLier* to notify when the outlier detection process finishes and *returnResultPatternClassification* to return the result of the

Figure 10 Class diagram of the outlier detection service implementation in the pattern module.

process. This class does not implement the clustering algorithm itself but incorporates a *PatternOutLier* object to which it delegates the processing. This solution decouples the module from the concrete implementation and enables the easy introduction of other implementations.

- *PatternOutLier*: this interface defines the structure so that new classes can implement the necessary methods to be a *PatternOutLier* object. This class defines the following methods: *inputInstance*, *isOutLier*,

runPattern, *returnResultPatternClassification*, *setAttributesOutLier* and *finishedRunsOutLier*.

- *ResultPatternClassificationOutLier*: this class contains the attributes with all the results of the outlier detection algorithms. A *ResultPatternClassificationOutLier* object can be sent to an application where the information about estimation may be used as clusters found with the algorithm and the maximum allowed distance of an object from the clusters centroid to not be considered an outlier.
- *PatternOutLierHadoopImpl*: this is a very important class in the outlier detection service, as it implements the *PatternOutLier* interface and all methods proposed for this interface. It implements the *k-means clustering algorithm* and calculates the radius of the each cluster to decide if an instance is an outlier or not. It implements the *k-means clustering algorithm* integrated with the *Big Data* processing instance of *Hadoop*. We use the *Mahout* framework implementation of the *k-means* algorithm [15].
- *SimpleKMeansOutLierHadoop*: this class implements the integration and runs the *Kmeans* thread in the *Hadoop* instance.

All algorithms' implementations can be changed as the object-oriented structured programming allows abstraction and decoupling. The abstraction is a very important characteristic to add other algorithms' implementations.

3.2 A Testbed Implementation

In this Section we describe the implemented testbed with a resource manager and client test application to validate the functionality of the proposed architecture.

Figure 11 shows the web page with status of the LinkSmart. It can be seen that the service, highlighted by a dashed line, called by *Classification-ManagerImpl* has started and was registered in the middleware with HID *0.0.0.7721126273323016844*.

It was developed a Servlet to visualise all *PHID* registered in the *Pattern Manager*. Figure 12 shows this servlet in execution. In the figure we can see the *HID* that has been assigned to the *Pattern Manager*, in this case the value *0.0.0.7721126273323016844*, we can also observe that there are 5 *PHIDs* registered in the *Pattern Manager* with their respective kinds of algorithms and number of instances inserted in the pattern recognition module.

The raw data used in the resource manager are from the Guildford Facility proposed in the European Smart Santander Project [14, 18].

HID	DESCRIPTION
0.0.0.759426008170215612	Des = OntologyManager: SID = OntologyManager PID = OntologyManager:
0.0.0.7721126273323016844	ClassificationManagerImpl
0.0.0.8986889843275488527	Desc = DataAcquisitionComponent SID = eu.linksmart.caf.daqc PID = DAqC:rayden
0.0.0.9086952901443329325	Des = QoSManager: SID = QoSManagerPort PID = QoSManager:
0.0.0.2077197722668330988	Des = TrustManager: SID = TrustManager PID = TrustManager:

Figure 11 The LinkSmart status web page.

The retrieved data were inserted in the Mysql [24] database and a class to simulate the resource manager was created. The resource manager provides temperature and light intensity values from a single sensor, called by *node25*.

The resource manager requests a *PHID* to register the new resource in the pattern manager with the parameter which identifies the requested service as outlier detection, value estimator or clustering service. Next, it starts to send the raw data to the pattern manager. Figure 13 shows the resource manager execution.

Figure 13 shows the application client that is created when the class is executed. In this execution the resource manager requests the service *ClassificationManagerImpl* by requesting a new *PHID*. It was assigned the *PHID 9167016986134* presented in Figure 12, and then it sends the raw data to the pattern manager. This data correspond to temperature and light intensity values obtained on 2014-02-01 between 00:00:00 and 23:59:59.

Figure 12 The pattern manager servlet status web page.

Figure 13 The resource manager execution.

The client application implements two functions: (i) it uses the pattern manager as a client; and (ii) uses the pattern manager as a coordinator to control execution of the algorithms. Figure 14 shows the execution of the client application.

Figure 14 The client application execution.

In Figure 14 it can be seen that the coordinator class requests and lists all resource managers registered in the pattern manager. Next, the class starts the execution of the clustering algorithm, and when the execution finishes the class receives the notification from the Event Manager, requests the classification of the new instance and then shows the cluster the instance belongs to, and the information about all clusters found in the execution. In this case, the instance was allocated in the cluster 4. The time, light intensity and temperature values of this instance are close to the values of cluster 4. It is also noteworthy that the 5 clusters that have been found express a consistent classification as far as the light intensity variation in a day. Figure 15 shows all existing instances in the test period plotted in a 3D graph.

In Figure 15 the clusters are identified by different colours. The x axis represents time that varies in the range 00:00:01 thru 23:59:59; the y axis represents light intensity that varies in range 0 thru 700; and the z axis represents temperature that varies in the range 16 thru 26 degrees Celsius. It can be seen that the algorithm created consistent clusters, separating the instances in 5 groups: 1st group represented in cyan, 2nd in blue, 3rd in magenta, 4th

Figure 15 All instances plotted in a 3D graph and clusters identified by different colors.

in green and the 5th in red. The centres of the black circles are the clusters' centroids found by the algorithm.

The model has been validated by the SSE (Sum of the Square Errors) and by the Silhouette coefficient, as suggested in [25]. To find the ideal number of clusters we minimise the SSE that however does not reach zero. We should plot the SSE decay curve and the ideal number of clusters is found at the curve's knee or diving point. The silhouette coefficient varies between 0 and 1 and the ideal number of clusters is the one closest to the Silhouette coefficient equal to 1. The obtained results for the SSE and the Silhouette coefficient are shown in Figure 16 as function of the number of clusters. According to the aforementioned criteria it is clearly seen that the ideal number of clusters is 5.

It is important to point out that it is out of the scope of this work the development of a specific IoT application or service. The main objective is to provide a contribution at the architectural level to enable customised data processing at a lower level layer than the application layer. It has been shown that the proposed architecture provides such functionality. In addition, the modular approach allows to different algorithms to be plugged in the architecture. The independence of the implemented algorithms from the original LinkSmart modules also allows them to be used in other platforms with any modification.

Figure 16 (left) - SSE versus the number of clusters; (right): silhouette coefficient versus number of clusters.

4 Conclusions and Future Work

In this work we have proposed a new IoT architecture that implements pattern recognition algorithms in middleware layer. It is based both on the IoT-A reference model and the LinkSmart middleware. Its scalability is ensured by the use of Big Data technology enabling physical objects and sensors to be directly plugged as resource manager classes. The object-oriented structured programming employed in the pattern manager allows other pattern recognition algorithms' implementations to be plugged in the future limiting the development tasks to the implementation of the interfaces proposed in the architecture.

The proposed architecture and its implementation contribute to enhance the use of the IoT LinkSmart middleware. This framework provides the scalability, contextualisation and flexibility enabling different kinds of devices to acquire environment context awareness. The information provided by a single light sensor, for example, can be read by various applications without any interference on each other. All sensors and devices present in a same environment are seen as in the same context, which is an important capability

in IoT and pervasive environments. In this way, several client applications are able to use the pattern recognition services. The raw data is processed only once in the middleware layer, so different applications may be simple and receive the contextualized information, without the need to process the original raw data. This approach reduces the network traffic and the overall energy consumption.

The Big Data implementation with Hadoop and Mahout provides great scalability allowing the creation of clusters with hundreds or thousands of Hadoop instances that can be plugged transparently into the LinkSmart and client applications.

The IPv6 protocol can be used in the communication between the resource manager, the LinkSmart middleware and client applications [22] which is an important feature to new IoT applications.

The testbed implementation validated the proposed architecture using real data from Smart Santander Project. The execution shows that the IoT-A architecture implementation, the LinkSmart middleware and the pattern recognition algorithms implemented in the middleware layer work perfectly.

As future work we intend to use the implemented algorithms with different real databases and contexts to understand user's behaviors, estimate lost parameters and detect outliers or erroneous values of the environment variables or context of the application. In addition, we plan to measure the energy savings and network traffic reduction indicated in this paper provided by the centralised processing of raw data in the middleware by the resource manager layer.

At last, we also plan to implement and validate feature extraction pattern recognition algorithms to create virtual sensors in the IoT architecture.

Acknowledgements

We acknowledge the ICT- 2009-257992 (SmartSantander) and the REDUCE project grant EP/I000232/1 under the Digital Economy Programme run by Research Councils UK that supported the development and deployment of the SmartCampus testbed.

References

- [1] J. R. de A. Amazonas. Network virtualization and cloud computing: Iot enabling technologies. *Casagras2 Academic Seminar*, September 2011.

- [2] M. Botts, G. Percivall, C. Reed, and J. Davidson. Ogc sensor web enablement: Overview and high level architecture. *Open Geospatial Consortium, Inc. Withpaper*, OGC 07-165, 2007.
- [3] EU FP7 Project CASAGRAS. Casagras final report: Rfid and the inclusive model for the internet of things. 2009.
- [4] D. et al. Clark. Making the world (of communications) as different place. *End-to-End Research Group, IRTF, ACM SIGCOMM Computer Communication Review*, pages 91–96, 2005.
- [5] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2 edition, November 2000.
- [6] Alexander Gluhak, Martin Bauer, Frederic Montagut, Vlad Stirbu, Matias Johansson, Jesus Bernat Vercher, and Mirko Presser. Towards an architecture for a real world internet. In *Towards the Future Internet - A European Research Perspective*, pages 313–324, 2009.
- [7] R.S. Hall, K. Pauls, and S. McCulloch. *OSGi in Action: Creating Modular Applications in Java*. In Action. Manning, 2011.
- [8] Y. Huang and G. Li. Descriptive models for internet of things. *International Conference on Intelligent Control and Information Processing*, August 2010.
- [9] Hydra. Service oriented architecture middleware for internet of things, 2007.
- [10] W. Joachim and Siemens Walewski. Internet of things architecture iot-a. *Deliverable D1.4 - Converged architectural reference model for the IoT v2.0*, 2012.
- [11] K. E. Kjaer. A survey of context-aware middleware. 2005.
- [12] Dajiang Lei, Qingsheng Zhu, Jun Chen, Hai Lin, and Peng Yang. Automatic k-means clustering algorithm for outlier detection. In Rongbo Zhu and Yan Ma, editors, *Information Engineering and Applications*, volume 154 of *Lecture Notes in Electrical Engineering*, pages 363–372. Springer London, 2012.
- [13] Kalle Lyytinen and Youngjin Yoo. Issues and challenges in ubiquitous computing. *Commun. ACM*, 45(12):62–65, December 2002.
- [14] M. Nati, A Gluhak, H. Abangar, and W. Headley. Smartcampus: A user-centric testbed for internet of things experimentation. In *Wireless Personal Multimedia Communications (WPMC), 2013 16th International Symposium on*, pages 1–6, June 2013.
- [15] Sean Owen, Robin Anil, Ted Dunning, and Ellen Friedman. *Mahout in Action*. Manning Publications Co., Greenwich, CT, USA, 2011.

- [16] R. Pamula, J.K. Deka, and S. Nandi. An outlier detection method based on clustering. In *Emerging Applications of Information Technology (EAIT), 2011 Second International Conference on*, pages 253–256, Feb 2011.
- [17] Hydra Project. Hydra project overview, June 2007.
- [18] Smart Santander FUTURE INTERNET RESEARCH and EXPERIMENTATION. Guildford facility. 2013.
- [19] G. Roussos. Sensor and actuators networks: from smart dust to the human internet. *Casagras2 Academic Seminar*, September 2011.
- [20] M. Sarnovsky, P. Kostelink, P. Butka, J. Hreno, and D. Lackova. First demonstrator of hydra middleware architecture for building automation. June 2005.
- [21] I.G. Smith. *The Internet of Things 2012: New Horizons*. CASAGRAS2, 2012.
- [22] Alberto M.C. Souza and Jose R.A. Amazonas. A novel smart home application using an internet of things middleware. In *Smart Objects, Systems and Technologies (SmartSysTech), Proceedings of 2013 European Conference on*, pages 1–7, 2013.
- [23] Helen Sun and Peter Heller. Oracle information architecture: An architect's guide to big data. In *An Oracle White Paper in Enterprise Architecture*, 2012.
- [24] S.M.M. Tahaghoghi and H.E. Williams. *Learning MySQL*. O'Reilly Media, 2006.
- [25] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.
- [26] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Fourth Edition*. Academic Press, 4th edition, 2008.
- [27] Sebastian Lange Thorsten Kramp, Rob van Kranenburg, editor. *Enabling things to talk: designing IoT solutions with the IoT architectural reference model*. Springer, Heidelberg, 2013.
- [28] D. Tracey and C. Sreenan. A holistic architecture for the internet of things, sensing services and big data. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 546–553, 2013.
- [29] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, January 1991.
- [30] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

- [31] I.H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2005.

Biographies

A. M. C. Souza. Graduated in Computer Science in 2005 from Cruzeiro do Sul University. Msc. (2010) with research area in complex networks from Instituto Tecnológico de Aeronáutica (ITA). He is currently Ph.D student at Escola Politécnica of the University of São Paulo. His research interests include Internet of Things and pattern recognition.

J. R. A. Amazonas. Graduated in electrical engineering (1979), and obtained his MSc (1983), Ph.D (1988) and Pos-doc (1996) degrees from Escola Politécnica of the University of São Paulo. He is associate professor at Escola Politécnica of the University of São Paulo and visiting scholar at the Technical University of Catalonia, Spain. Prof. Amazonas acts as referee of the journals: IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Education, IEEE Transactions on Computers, Elsevier Computer Networks.