Journal of Universal Computer Science, vol. 24, no. 10 (2018), 1452-1471
submitted: 4/3/18, accepted: 27/6/18, appeared: 28/10/18 © *J.UCS*

An Architecture for IoT Management Targeted to Context Awareness of Ubiquitous Applications

Rodrigo Souza, João Lopes, Cláudio Geyer
(Federal University of Rio Grande do Sul, Porto Alegre - RS, Brazil
{rssouza, jlblopes, geyer}@inf.ufrgs.br)

Anderson Cardozo, Adenauer Yamin
(Federal University of Pelotas, Pelotas - RS, Brazil
kledac@gmail.com.br, adenauer@inf.ufpel.edu.br)

Jorge Barbosa
(University of the Vale do Rio dos Sinos, São Leopoldo - RS, Brazil
jbarbosa@unisinos.br)

Abstract: The recent advances in the Internet of Things (IoT), which has provided increasing availability of networked sensors and actuators, have given context awareness research in the UbiComp area a new perspective. In this sense, the main contribution of this paper is the proposition of a distributed architecture for IoT, called CoIoT (Context awareness in the Internet of Things). This architecture is designed to provide proactive management of the interactions with the physical environment. To evaluate the functionalities of the proposed architecture we implemented a case study in the agricultural area, specifically in the monitoring of seed analysis laboratory.
Key Words: Ubiquitous Computing, Context Awareness, Internet of Things, Middleware
Category: L.7, C.2.4, J.7

1 Introduction

In Ubiquitous Computing (UbiComp), computational systems must be able to react to changes in the state of different contextual variables of interest. These contextual variables should be collected in highly distributed environments [Knappmeyer et al. 2013] [Li et al. 2015]. Meanwhile, scientific advances and technological developments in the field of IoT have enabled the use of large-scale sensors, which are sources of contextual information for context-aware ubiquitous applications [Perera et al. 2014].

Several research challenges related to the use of IoT to obtain contextual information are associated with the differences between the high level of ubiquitous applications requirements and the management tasks for IoT devices, which are related to the electronic characteristics involved [Gubbi et al. 2013] [Perera et al. 2014].

The main contribution of this paper is to fill this gap by proposing CoIoT (Context awareness in the Internet of Things), an architecture integrated to

EXEHDA Middleware (Execution Environment for Highly Distributed Applications) [Lopes et al. 2014a], capable to provide support to the processing of sensors and actuators. EXEHDA provides software architecture based on services that aim to create and manage a ubiquitous environment as well as the running applications on this environment.

CoIoT is an event-based architecture managed by rules, which provides a distributed processing environment, being able to act proactively in the collection of contextual information of the physical environment as well as remotely perform operations on it.

CoIoT differs from other architectures identified in the literature by integrating in the same architecture functionalities that treat context awareness as well as manage the IoT infrastructures. In addition, CoIoT proposes a distributed architecture for context processing that enable it to perform this processing near where the contextual data are generated.

This article is organized as follows: Section 2 presents the modeling of CoIoT detailing its architecture and features. In section 3, the prototype is presented and the tests are carried out in the area of agriculture. Related work is presented in section 4. Finally, the final considerations of this article are held in section 5.

2 Background

2.1 Internet of Things

The Internet of Things (IoT) is a computational approach that has been highlighted mainly due to the interest of technology companies. Because it is a new area, it has many open research problems, stimulating the involvement of the scientific community. IoT has also been considered as an approach to promote the ubiquity of computing solutions in the real world, mainly to overcome the lack of infrastructural aspects of UbiComp [Caceres and Friday 2012].

IoT points to a scenario in which smart objects or things (such as sensors and actuators) have the ability to communicate and transfer data over the Internet with minimal human intervention. With this feature, IoT can be considered an essential element to consolidate the integration between the computational systems and the physical environment, with the potential to produce large amounts of contextual information [Atzori et al. 2010] [Perera et al. 2014].

In the Internet of Things, computational devices are widely distributed in the environment, where they collect contextual information and generate actuations. Applications that use the information produced need in many cases to make decisions quickly and reliably.

One of the central features of the Things of Internet refers to the great heterogeneity of physical devices that may be present in the environment. This heterogeneity concerns the different capabilities and technological resources available in each device, whether software or hardware.

In the IoT the number of devices can be very high. Thus, it is not desirable for ubiquitous application developers to engage in the treatment of low-level technological aspects. Therefore, middleware solutions must treat heterogeneity and allow interoperability and integration of the devices and services that are part of the environment. The challenge is to manage the low-level aspects relevant to IoT devices, while meeting the high-level demands of UbiComp applications [Delicato et al. 2013].

2.2 Event Handling

In the Internet of Things, environmental events occur when there is a major change in a context of interest, for example, a temperature reaching certain value or the identification of a user entrance into a room and much more. These events should be intercepted by the management system and notifications should be sent to the applications so that they can provide the appropriate treatment for those data [Perera et al. 2014].

IoT environments have the potential to generate many events that must be managed by the underlying architecture. This management enables to handle events when they happen, allowing a quick response whenever necessary [Razzaque et al. 2016].

Events are frequently identified as primitives (discrete) or complex (composite). A primitive event refers to an instantaneous atomic occurrence of an event of interest at a given time, while a complex event (also called composed event) is the combination of primitive events in a given time interval [Terfloth 2009].

2.3 EXEHDA Middleware

The foundation of our proposal is the EXEHDA Software Architecture [Yamin et al. 2005] [Lopes et al. 2014a], in which cells form a large-scale computing environment. These cells are composed of several mobile and stationary physical resources. The components of the computing environment, such as: data, codes, devices, services and resources, are ubiquitous and managed by a middleware that provides continuous access to them.

The EXEHDA middleware software architecture, which is shown in Figure 1, aims to provide an integrated solution to build and execute large-scale ubiquitous applications. The execution of such applications is supported by EXEHDA middleware.

EXEHDA architecture is divided into three layers, logically organized: (Upper) application layer; (Middle) support layer, and execution environment; and (Lower) basic systems' layer. The Upper Layer corresponds to the abstractions

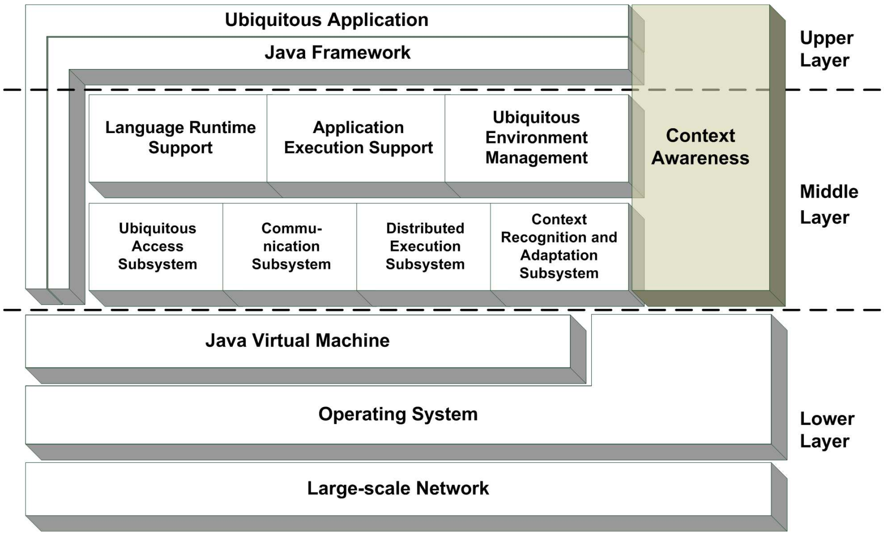

Figure 1: EXEHDA Middleware Software Architecture

provided to the application designer to ease the development of context-aware application. This is mainly obtained by the provision of a Java Framework. In this layer and the next, we also have the representation of context awareness. The objective behind this is to underscore its importance in the architecture, highlighting their presence in the design of many components.

The Middle Layer contains the support mechanisms for the implementation of ubiquitous applications and adaptation strategies. This layer consists of two levels: the first level consists of the application service modules, and the second level is formed by the EXEHDA basic services. These basic services enable features required for the upper level and cover various aspects, such as ubiquitous access, communication, distributed execution, context recognition, and adaptation.

Finally, the Lower Layer of the architecture is composed of native languages and systems that integrate the execution in the physical infrastructure. For reasons of portability, in this layer the platform for implementation is the Java Virtual Machine, which is available for different operating systems. The architecture assumes the existence of a network to support the execution of components and services on a global scale.

EXEHDA is a service-oriented architecture that contributes in three perspectives: (i) it provides a management through services to control the physical environment in which the processing will take place; (ii) it supports the execution of applications, by providing the services and abstractions needed to implement the follow-me semantics; and (iii) offers an API (Application Programming Interface) to foster ubiquitous application development.

The operational requirements in a highly heterogeneous environment, in which hardware capabilities and software availability on each device may vary, have motivated the use of pluggable services. In this approach, the middleware minimum core extends its functionalities according to the availability of resources. The loading of these pluggable services is done on-demand and, moreover, is context adaptive. In this way, we are able to employ implementations of services that are better tuned to each device and also reduce resource consumption by only loading services that are effectively used. Such scheme is possible because services are defined by their semantics and interface rather than by a specific implementation.

EXEHDA is composed of several integrated services. These services are conceptually organized in subsystems: data and code ubiquitous access, uncoupled spatial and temporal communication, large-scale distribution, context recognition and adaptation.

Regarding the management of context information, EXEHDA provides the following services that constitute context recognition and adaptation subsystem.

The Monitor Service implements a monitoring scheme based on sensors, which employs indexes to describe specific aspects of the environment. These sensors can be customized through parameters. The whole set of sensors installed on a node is part of the node description information registered in the CIB Service. The data generated by each sensor is gathered by the Monitor Service, which typically runs on the same node in which sensors are installed. The gathered data are published by the Monitor Service to a Collector Service, which typically runs on the base-node.

Both the gathering of data by the sensors and the publication to the Collector Service by the Monitor occur in discrete multiples of a per-node configured *quantum*. The quantum parameter allows the resource owner to control, externally to the middleware, the degree of intrusion of the monitoring mechanism in the host. After a *quantum* of time expires, the Monitor Service executes a pooling operation over the active sensors in the node. Then, it applies the publishing criteria specified for the sensor data, determining, or not, the generation of a publishing event for that sensor. Thus, the events generated after a quantum expiration are grouped into a single message, reducing the amount of data that the Monitor has to transmit to the Collector.

The Collector Service aggregates information from several monitors in the cell and forwards them to the registered consumers. Among such consumers are other middleware services like the Context Manager. The Context Manager service is responsible for the processing of the raw information obtained by the monitoring, producing abstract information concerning the contexts of interest.

3 CoIoT: Design and Modeling

The architecture of CoIoT was proposed in order to enable the EXEHDA to continue providing context awareness in view of the challenges of the IoT, where the acquisition of context information is done from a large number of heterogeneous devices, dynamically distributed and restricted capacity, it was proposed the architecture of CoIoT[1].

Considering this motivation, CoIoT proposes a computational environment composed of executing cells, in which computing devices are distributed (see Figure 2). Each cell consists of the following components:

- EXEHDAbase, the central element of the cell, being responsible for all basic services and constituting a reference for the other elements.
- EXEHDAnode, which corresponds to computing devices, responsible for running the applications.
- EXEHDAnode mobile, a subcase of EXEHDAnode, corresponding to typical mobile devices that can move between cells of the ubiquitous environment, such as laptops, tablets or smartphones.
- EXEHDAedge, responsible for the interoperability between the middleware services and various types of gateways.
- EXEHDAgateway, which is the element responsible for sectoring the data collection points and/or distributed operations available in the physical environment, performing their interaction of these with the other middleware components.

The approach to context treatment proposed on CoIoT has its functionality distributed between two types of servers: Context Server (EXEHDAbase) and Edge Server (EXEHDAedge). The Edge Server is designed to work primarily on the management of interactions with the physical environment. Meanwhile, the Context Server operates in the storage and processing of contextual information [Lopes et al. 2014b].

In the CoIoT proposal the premise is that the sensors and/or actuators are incorporated into the Edge Server through gateways. Gateways are then used to treat both hardware and software heterogeneity, typical to sensing and/or acting devices, performing protocol conversion and management of devices in addition to providing these communication capabilities via Internet.

CoIoT enables the identification and automatic discovery of sensing devices and/or actions from gateways integrated into the architecture through the UPnP (Universal Plug and Play) [UPnP 2016] communication protocol.

[1] CoIoT-EXEHDA Source Code: http:amplus.ufpel.edu.br/coiot

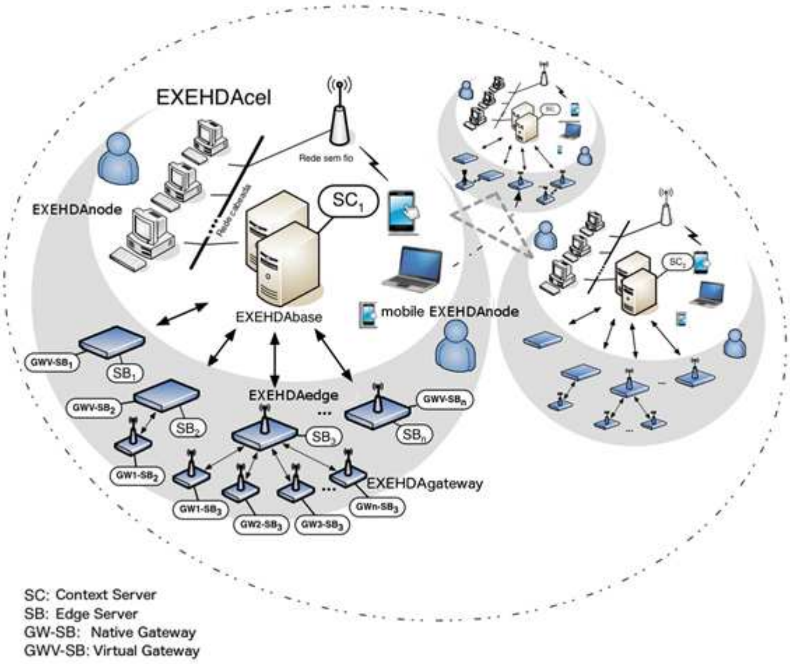

Figure 2: Ubiquitous Environment

The discovery features and auto-configuration of devices explores the UPnP protocol. This protocol is a facilitator when the addition and/or removal of devices in scalable and dynamic environments. Coupled with the fact that is possible it's use with consolidated protocols and Internet standards such as HTTP, SOAP and XML, makes UPnP widely used in the setting of IoT devices.

The CoIoT provides support for primitive and complex events, which can be used to trigger ECA (event-condition-action) [Terfloth 2009] rules. The event handling model proposed for CoIoT considers a set of primitive events generated from (i) changes of state of the contexts of interest collected through sensors; (ii) activation/deactivation of actuators; and (iii) changes in the infrastructure of the computing environment. These events are shown in Table 1. CoIoT supports complex events through event composition using conditional logic treated by ECA rules and processed by the Context Processors.

3.1 Architectural Model

The proposed architecture for CoIoT, shown in Figure 3, has been designed with the objective to manage different IoT devices, e.g. heterogeneous sensors

Table 1: Primitive Events

Event	Level	Description
Publication	Gateway/ Edge Server	Occurs when a contextual data is sent to the Edge Server or the Context Server
Actuation	Gateway	Occurs when the actuator is activated
NewDevice	Gateway	Occurs when a new sensor/actuator is connected
DeviceDisconect	Gateway	Occurs when a sensor/actuator is disconected
NewGateway	Edge Server	Occurs when a new gateway is conected
GatewayDisconect	Edge Server	Occurs when a new gateway is disconected

and actuators. This architecture is premised upon acting autonomously both in the collection and processing of contextual information as on acting over the physical environment, as these activities continue to be performed even in periods in which applications interested in its use are inoperative.

The context processing in the CoIoT is performed in a distributed manner between the Edge Servers and the Context Server. The Rule Engine module (Edge Server) is the first processing level, while the Context Processor (Context Server) is the second level.

The rules submitted to the Rules Engine should be developed to serve, primarily, critical events; the treatment should be performed in the shortest possible time and with minimal faults. This is due to the fact that the Edge Server is usually allocated physically close to the monitored environment, allowing for acting (warning, activation/deactivation of electromechanical equipment, etc.) regardless of a possible loss of communication with the Context Server in result of a network failure [Cardozo et al. 2016]. On the other hand, rules that require dealing with the treatment of historical information, access data collected from other Edge Servers, or needing a different model of contextual processing must be processed in the Context Server.

Both context-processing modules are designed based on the model ECA which can be triggered from events produced by the environment. Although the ECA model is the basic mechanism of contextual treatment being used, both the condition being treated and the action to be performed by the rule admit other processing models that can be invoked by the rule, which are due to the type of domain application to be served by CoIoT.

The Contexts Repository module uses a relational model for the representation of contextual information, which provides a historical record of those data. The structure of the Context Repository reflects the architectural organization of the Middleware EXEHDA thus contemplating the relationship between applications, components, sensors, environments and contexts of interest. The repository also stores the architecture configuration data and publications of existing sensors in the ubiquitous environment. These data are used by the Context processor module to trigger the appropriate actions depending on the contextual

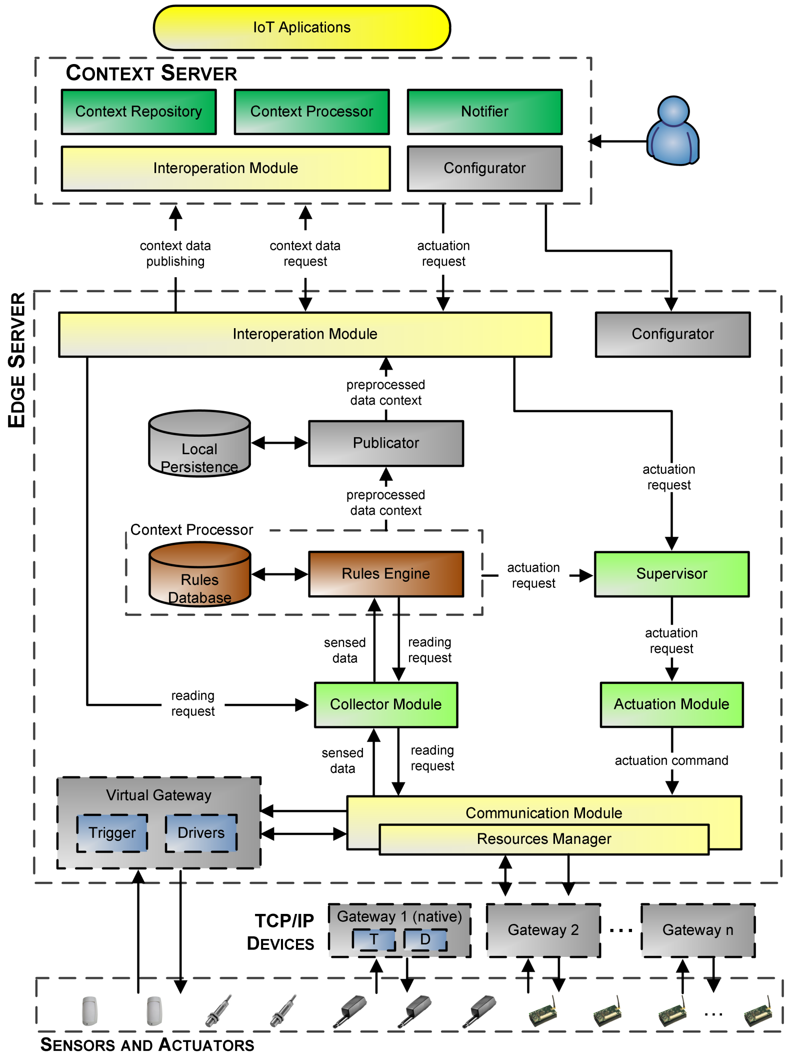

Figure 3: CoIoT Architecture

information.

Given the inherently distributed characteristic of ubiquitous applications, Interoperation Modules of CoIoT have been designed to promote interoperability between the Edge Servers and Context Server, as well as with other middleware services. The design of this module was based on the REST architectural style [Fielding 2000].

The Notifier Module has the function of generating notifications from the context processing results carried out by the Context Processor. This module uses a notification strategy based on the model publisher/subscriber, which receives subscriptions for all services and/or applications that require notifications regarding changes in the states of context.

All the settings required for the operation of CoIoT are managed through a web interface provided by the Configurator Module. Among the features offered we have: the configuration of sensors and actuators (adding, removing and changing), the management of device drivers, management of contextual processing rules, configuration of access to Context Server and Edge Server, Gateways management, among others.

The Publisher module has the function of request of sending contextual information to the other layers of the middleware, interoperating with Context Server via the Interoperation Module. The publications are organized in a FIFO system and are processed according to network availability. Considering the possible miscommunications between the Edge Server and the Context Server, a Local Persistence Module for Edge Server has been designed to perform temporary storage queue of contextual information until these are propagated to the context server.

With the purpose of ensuring interoperability with market technologies, and also enhance the distribution of gathering initiatives and/or acting, two types of gateways were used: (i) Proprietary Gateway, which has heterogeneous features varying according to their manufacturers; and (ii) Native Gateway, whose features are integrated into the CoIoT architecture. The Virtual Gateway module acts as a virtualization of the Native Gateways and implements two basic types of modules: Drivers and Triggers. Drivers are architectural modules responsible to promote the access to the the sensors, as well as for the execution of commands sent to the actuators. Drivers encapsulate and control the sensors and actuators in an individualized manner, preventing operational differences of these devices to propagate to other components of the architecture. Triggers manage the reading sensors through events and have been designed to handle the two main types of events to be treated in the IoT: temporal events and environmental events.

The Communication Module and the Resource Manager have been designed to manage aspects associated with communication between the gateways and the Edge Server. The Communication Module manages the communications via REST API as the Resource Manager provides a discovery engine that manages the connection and disconnection of devices on the network, typical occurrences of IoT.

The Collector Module has the function of directing the gathering requests to the respective gateways in accordance with instructions from the Rules Engine, the Context Server, or applications. The Supervisor module brings together the actuation commands, receiving the control parameters and resolving any conflicts between the requests from different sources. The Actuator module has a similar function to the Collector Module, receiving the actuation commands and the operating parameters (length, activation energy...) and forwarding then to the appropriate gateways for further processing.

4 CoIoT: Prototyping and Testing

This section summarizes the main aspects of prototyping and testing by the AMPLUS project (Automatic Programmable Monitoring and Logging Ubiquitous System), which was used to evaluate the functionality of CoIoT. The case study includes tasks related to the sensing, collecting, processing and notification of events detected in the physical environment. In this case study a tool was developed to assess the main features of the proposed architecture.

The AMPLUS project is designed to provide mobile services, context aware services that allow the storage of contextual conditions that characterize the equipments of the Didactic Laboratory Seed Analysis - Federal University of Pelotas (LDAS-UFPel - http://amplus.ufpel.edu.br/ldas) as well as the generation of notifications and autonomous actions when necessary. LDAS has been used mainly by PhD students of the agriculture area.

4.1 Hardware Infrastructure

To evaluate the features of CoIoT, a set of devices that consists of a native Gateway, 15 sensors and one actuator were used in the LDAS. The sensors selected for this case study are based on the 1-Wire technology [2]. This technology is characterized as a data transmission network based on addressable electronic devices, and stands out for its versatility and ease of implementation.

The kind of temperature sensor used can be seen in Figure 4 (D). This sensor is wrapped in an aluminum casing for give greater strength and isolation from moisture. The Native Gateway (Figure 4 (A)) was developed using NodeMCU. NodeMCU can be connected to up to seven devices 1-wire. NodeMCU [3] is an open source platform for the development of IoT devices (Figure 4 (B)). To explore the reactive characteristic of the architecture, an actuator (warning light) based on the 1-wire technology was also used. This actuator is triggered when the attention of laboratory workers is required with some equipment.

4.2 Software Infrastructure: Main Characteristics

The majority of the CoIoT Edge Server prototype was written in Python using a Raspbian operating system. The hardware used on the Edge Server is a Rasp-

[2] http://www.maximintegrated.com
[3] http://nodemcu.com/

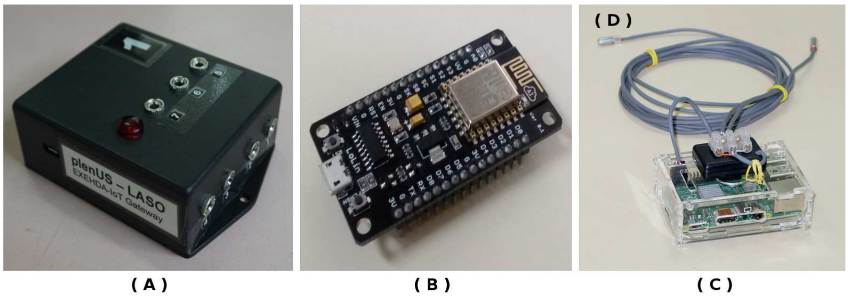

Figure 4: (A) Native Gateway; (B) NodeMCU; (C) Raspberry PI; (D) Temperature sensor DS18B20

berry Pi [4] (see Figure 4 (C)). The Context Server is installed on a computer with an Intel dual-core E3400-2.6GHz processor with 4GB of memory, with a Linux Ubuntu Server operating system. We used Drools[5] to implement the Rules Engine and the Context Processor.

The reading of the sensors is accomplished through specific drivers that perform an individualized treatment of devices according to the technical characteristics of each one, thus abstracting the technological differences between them. The Interoperation Module was developed using Sails.js [6], an MVC framework (Model-View-Controller) directed to the Node.js programming language [7]. The developed REST API provides resources to deal with the sensors and actuators, as well as to carry out the publication of the data collected. Data sent through the REST operations are structured in JSON data model.

4.3 Software Infrastructure: Solutions Developed for the LDAS

CoIoT supports the operation of the evaluation scenario through triggers that fire read requests for the sensors, and a set of contextual processing rules. Triggers are used to manage the temperature sensors reads of BODs Incubators (Biochemical oxygen demand) in two situations: (i) at regular time intervals; and (ii) when the value is outside a specified range. The BODs Incubators are used in LDAS to conduct seed germination tests, which require precision regarding specific limits for temperature variation.

The rules used for the context processing were organized between Edge Servers and the Context Server in order to attend the proposed scenario. The criteria used for the distribution rules are: (i) to minimizing the data stream;

[4] http://www.raspberrypi.org
[5] http://www.drools.org/
[6] http://sailsjs.org/
[7] https://nodejs.org

and (ii) to continue monitoring even in times of loss of communication between servers.

The rules used are shown in Tables 2 and 3.

Table 2: Edge Server Rules

Rule Name	Event	Condition	Action
Read Temperature	ReadSensor	If value outside a specified range	Triggers warning light
Publish Temperature	Publication	-	Publishes temperature to the Context Server

Table 3: Context Server Rules

Rule Name	Event	Condition	Action
Read Temperature	ReadSensor	If value outside a specified range	Notifies user (SMS/e-mail)
Persists Temperature	Publication	-	Persists temperature in the Context Repository

We used distributed ECA rules to operationalize this evaluation scenario. Figure 5 shows the form of processing of these rules, using a BPMN notation.

The developed tool enables the selection of the context of interest to be displayed, which can be in the form of a text report or through graphical mode. Through the LDAS tool, a researcher can have access to the visualization of changes in temperature and humidity values occurred in BODs Incubators during periods of analysis, which directly influence the results of the germination process of the seeds.

The management application developed for the Context Server is responsible for performing the four basic operations used in relational databases (CRUD - Create, Read, Update and Delete), that is, it is able to register, read, update, and remove data from the Context Information Directory.

The registration or removal by the application of devices for sensing and actuation can be performed either automatically, in the case of devices that support the UPnP protocol, or manually. It is worth remembering that the automatic registration will only be finalized through authentication by the user, thus avoiding the registration of unrecognized devices.

Access to the management application only happens through login, and each user can have a different level of access to the menus according to his permission. In addition to enabling operations in the database, the application is responsible for creating an intuitive interface so that the user can manage their resources in

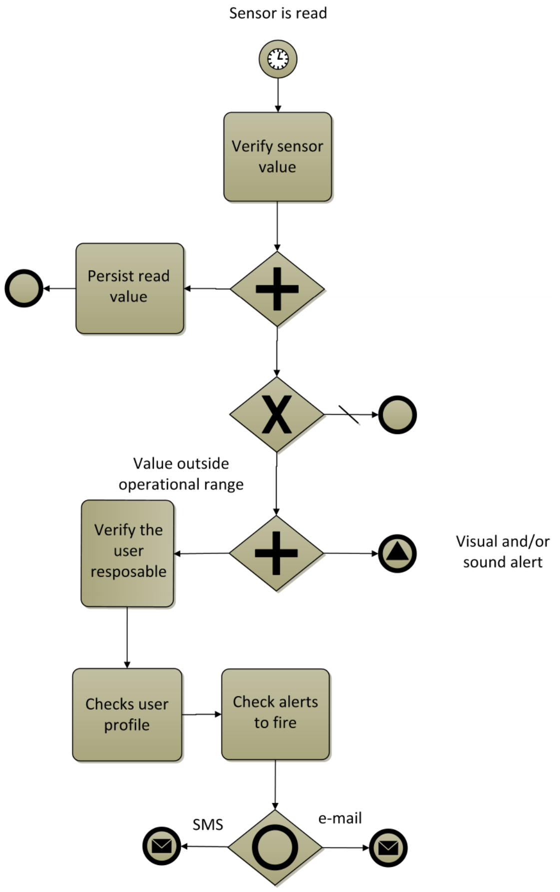

Figure 5: ECA Rule processing

a simple way. Considering these aspects, some functionalities have been created and arranged in a menu (see Figure 6).

Through the menu the user is able to change all configuration regarding the Context Server, Edge Servers and Gateways, as well as manage the contextual processing rules specific to these devices. It is also possible to manage all types of sensors and actuators used in the environment, and also configure reading operations schedules and actuations.

The application in its visualization mode allows the selection of one or more sensors, within a context of interest, to display the historical record of its contextual information. This information can be presented in the form of a textual report (see Figure 7 or through graphical mode (see Figure 8). The selection of sensors to be displayed is made from a menu that supports multiple selections.

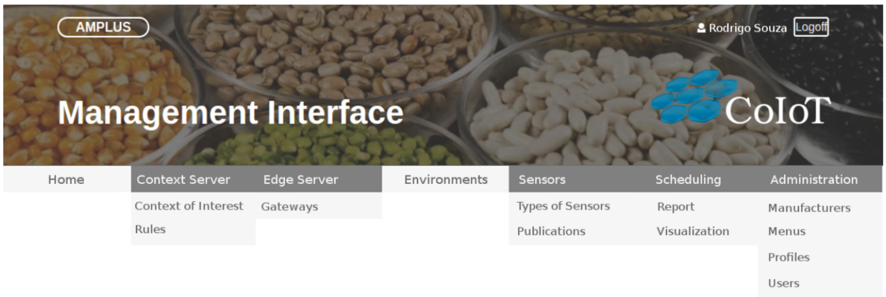

Figure 6: Management application

Also available is an inspection feature that allows the comparison of values at a given moment of time. The data time window being displayed can be defined by the user through the same graphical user interface that displays the sensed values.

In order to promote proactivity of AMPLUS with the community of users, interfaces have been developed for communication services, as e-mail and SMS. The Context Server produces these messages from the processing of contextual rules autonomously.

The routine of the laboratory technicians implies in mobility in the different rooms of the LDAS. To address this situation, an interface was developed to provide visual warnings, which are activated whenever a device is in a state that requires attention. Considering these alerts, details can be inferred through the visualization tool interface developed for the AMPLUS Project.

5 CoIoT: Evaluation

This section presents the experiment details and the results obtained with the evaluation of the application.

According to [Knappmeyer et al. 2013], the necessary transparency due to the autonomous operation of middleware in context-aware applications introduces a difficulty in the evaluation of the proposed functionalities. In this sense, one of the main strategies employed to evaluate middleware is the use of applications in which users' experiences can be explored explicitly or implicitly.

The user's experience with the application can be assessed explicitly, through the use of interviews and questionnaires, or implicitly through observations. This strategy can directly reflect the usability of the applications, indicating the ability of the architecture to meet the requirements of the applications, as well as enabling the verification of the correct functioning of its modules [Knappmeyer et al. 2013].

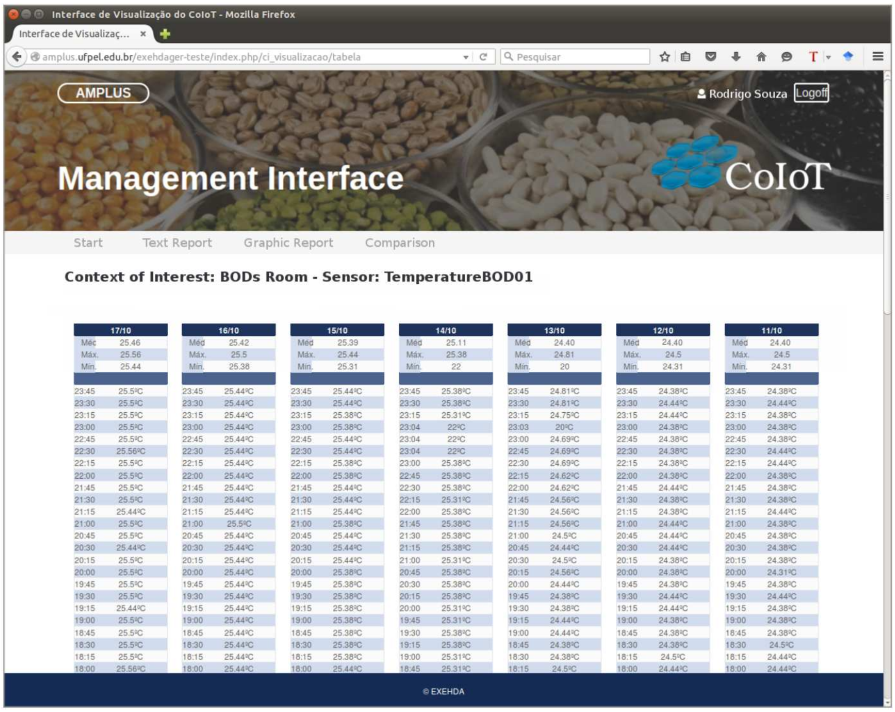

Figure 7: Textual Report

The evaluation regards the application acceptance by the user community, and involved LDAS volunteers. For the study we considered 4 researches, 5 students, and 1 technician. Each participant used a desktop. We performed a basic training on the application operation beforehand. Participants were asked to use the application and respond to an evaluation questionnaire regarding their experience using the system.

The answers should be within a range of five points: 1 point (totally disagree) to 5 points (totally agree). To evaluate the acceptability of the model and check the system usability, the questionnaires were defined based on the Technology Acceptance Model (TAM) [Yoon and Kim 2007]. The TAM model considers the following main themes for application acceptance: (i) Ease of Use: means the degree in which users evaluate the application may reduce their effort; (ii) Usefulness: means the degree in which users evaluate the application may improve their performance.

Table 4 and Table 5 contain the questionnaire applied to users, and the answers obtained. The questions were designed in order to be simple, short and direct. Both tables present in the first column the question and the percentage

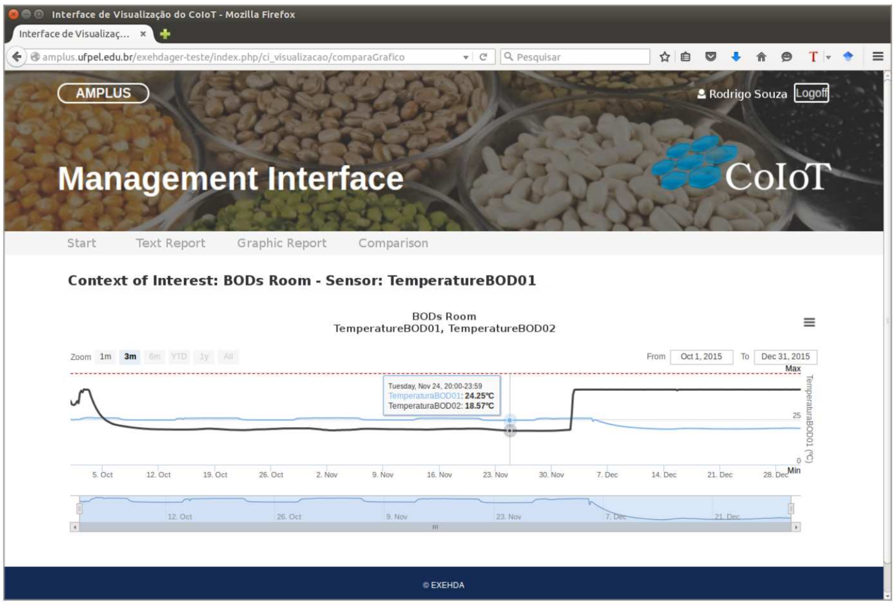

Figure 8: Graphical Report

obtained, with the number of users in brackets, ranging from "Totally Disagree" to "Totally Agree". The last column shows the consolidation average percentage score obtained by the responses, which varied between zero and five.

The results analysis shows that the approval is high for both ease of use, as for usefulness. However, there were results rated "indifferent" in the last two issues of usefulness. This can be interpreted as a concern for the quality control of experiments developed in LDAS, which depends basically on the use of autonomic mechanisms, without the usual human intervention, for issuing alerts for contextual states that require immediate actuation. In this case, a strategy that can be adopted is to intensify the testing and validation with users and start a gradual deployment of applications.

6 Related Work

The study of literature has identified related work, from which the following were selected: EcoDiF [Pires et al. 2014], Xively [Xively 2016], Carriots [Carriots 2016] and LinkSmart [Kostelnik et al. 2011]. The aspects considered in the selection of these related studies were: (i) heterogeneity support; (ii) event management support; (iii) context awareness; and (iv) interoperability.

Among the related work, context awareness is supported only by LinkSmart, but the support offered is limited. Meanwhile, CoIoT provides mechanisms to

Table 4: Ease of Use Evaluation

Question	**Totally Disagree**	**Partially Disagree**	**Neutral**	**Partially Agree**	**Totally Agree**	**Average**
1. The application easy to understand.	0,0%(0)	0,0%(0)	0,0%(0)	40,0%(4)	60,0%(6)	4,6
2. The application is easy to use.	0,0%(0)	0,0%(0)	0,0%(0)	30,0%(3)	70,0%(7)	4,7
3. The option are clear and objectives	0,0%(0)	0,0%(0)	10,0%(1)	20,0%(2)	70,0%(7)	4,6
4. With little effort I can select a context of interest.	0,0%(0)	0,0%(0)	0,0%(0)	20,0%(2)	80,0%(8)	4,8
5. The application interface is properly adapted to the devices.	0,0%(0)	0,0%(0)	0,0%(0)	30,0%(3)	70,0%(7)	4,7

Table 5: Usefulness Evaluation

Question	**Totally Disagree**	**Partially Disagree**	**Neutral**	**Partially Agree**	**Totally Agree**	**Average**
1. The options presented are relevant.	0,0%(0)	0,0%(0)	0,0%(0)	30,0%(3)	70,0%(7)	4,7
2. The application makes it easy to obtain contextual data from multiple sensors.	0,0%(0)	0,0%(0)	0,0%(0)	40,0%(4)	60,0%(6)	4,6
3. The application facilitates immediate action from the issuance of an alert or message.	0,0%(0)	0,0%(0)	30,0%(3)	30,0%(3)	40,0%(4)	4,1
4. I would use this application in my work?	0,0%(0)	0,0%(0)	30,0%(3)	20,0%(2)	50,0%(5)	4,2

perform the collection and processing of contextual data in a distributed manner through a rules management, as well as through methods for performing actions on the physical environment.

All related researches present triggers strategies to manage the data flow transmitted between the different devices involved. A smaller data flow has benefits, especially with regard to scalability and power consumption. However, Xively and Carriots do not offer triggers associated with gathering. In CoIoT the triggers approach is designed to allow the customization of data collection through events considering the physical variability characteristics of each monitored variable, which provides a minimization of data flow between the gateways and the Edge Server.

Event handling is supported by all the selected related works, but only Carriots uses rules in this treatment. In addition, the distributed management rules between Context and Edge Servers are a differential in relation to other projects. This connection processing functionality in all related works is usually restricted to a single device.

7 Conclusion

This article summarizes the research efforts associated with the design of CoIoT. CoIoT is an architecture for the Internet of Things, integrated with middleware EXEHDA, which manages the collection and pre-processing of contextual information, supporting the operations in the environment.

The main contribution of this work is to designing an architecture for IoT directed to context awareness. The proposal is an event oriented architecture, rules based and able to manage the collection and processing of contextual information in a distributed infrastructure. The strategy adopted for CoIoT expanded the scope of use of Middleware EXEHDA, allowing its use in the autonomous management of IoT resources, which minimizes the user interference.

The following aspects should be considered in future works of the research: (i) expanding the use of CoIoT in LDAS, enabling the monitoring of other laboratory equipment and, consequently, incorporating other types of sensors and actuators; and (ii) continuing the integration procedures of CoIoT with the different services and features of Middleware EXEHDA.

References

[Atzori et al. 2010] Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805.

[Caceres and Friday 2012] Caceres, R. and Friday, A. (2012). Ubicomp systems at 20: Progress, opportunities, and challenges. *Pervasive Computing, IEEE*, 11(1):14–21.

[Cardozo et al. 2016] Cardozo, A., Yamin, A., Xavier, L., Souza, R., Lopes, J., and Geyer, C. (2016). An architecture proposal to distributed sensing in internet of things. In *2016 1st International Symposium on Instrumentation Systems, Circuits and Transducers (INSCIT)*, pages 67–72.

[Carriots 2016] Carriots (2016). Carriots iot platform. Accessed on January 2016.

[Delicato et al. 2013] Delicato, F. C., Pires, P. F., and Batista, T. V. (2013). *Middleware Solutions for the Internet of Things*. Springer Briefs in Computer Science. Springer.

[Fielding 2000] Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Phd thesis, University of California, California-USA.

[Gubbi et al. 2013] Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660.

[Knappmeyer et al. 2013] Knappmeyer, M., Kiani, S., Reetz, E., Baker, N., and Tonjes, R. (2013). Survey of context provisioning middleware. *Communications Surveys Tutorials, IEEE*, 15(3):1492–1519.

[Kostelnik et al. 2011] Kostelnik, P., Sarnovsky, M., and Furdik, K. (2011). The semantic middleware for networked embedded systems applied in the internet of things and services domain. *Scalable Computing: Practice and Experience*, 12(3).

[Li et al. 2015] Li, X., Eckert, M., Martinez, J.-F., and Rubio, G. (2015). Context aware middleware architectures: Survey and challenges. *Sensors*, 15(8):20570.

[Lopes et al. 2014a] Lopes, J., Souza, R., Geyer, C., Costa, C., Barbosa, J., Pernas, A., and Yamin, A. (2014a). A middleware architecture for dynamic adaptation in ubiquitous computing. *Journal of Universal Computer Science*, 20(9):1327–1351.

[Lopes et al. 2014b] Lopes, J., Souza, R., Pernas, A., Yamin, A., and Geyer, C. (2014b). A distributed architecture for supporting context-aware applications in ubicomp. In *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, pages 584–590.

[Perera et al. 2014] Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2014). Context aware computing for the internet of things: A survey. *IEEE Communications Surveys and Tutorials*, 16(1):414 – 454.

[Pires et al. 2014] Pires, P. F., Cavalcante, E., Barros, T., Delicato, F. C., Batista, T., and Costa, B. (2014). A platform for integrating physical devices in the internet of things. *Proceedings of the 12th IEEE International Conference on Embedded and Ubiquitous Computing*, pages 234–241.

[Razzaque et al. 2016] Razzaque, M. A., Milojevic-Jevric, M., Palade, A., and Clarke, S. (2016). Middleware for internet of things: a survey. *Internet of Things Journal, IEEE*, 3(1):70–95.

[Terfloth 2009] Terfloth, K. (2009). *A Rule-Based Programming Model for Wireless Sensor Networks*. Phd thesis, Freie Universita, Berlin-Germany.

[UPnP 2016] UPnP (2016). Upnp resources. Accessed on January 2016.

[Xively 2016] Xively (2016). Xively iot application platform. Accessed on January 2016.

[Yamin et al. 2005] Yamin, A. C., Augustin, I., Barbosa, J., da Silva, L. C., Real, R. A., Filho, A. S., and Geyer, C. F. R. (2005). Exehda: Adaptive middleware for building a pervasive grid environment. *Frontiers in Artificial Intelligence and Applications - Self-Organization and Autonomic Informatics*, 135:203–219.

[Yoon and Kim 2007] Yoon, C. and Kim, S. (2007). Convenience and TAM in a ubiquitous computing environment: The case of wireless LAN. *Electronic Commerce Research and Applications*, 6(1):102–112.