A Sensing-as-a-Service Context-Aware System for Internet of Things Environments

Everton de Matos, Leonardo Albernaz Amaral, Ramão Tiago Tiburski,
Matheus Crespi Schenfeld, Dario F. G. de Azevedo, Fabiano Hessel
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Porto Alegre - RS - Brazil
(everton.matos.001, leonardo.amaral, ramao.tiburski, matheus.schenfeld)@acad.pucrs.br, (dario, fabiano.hessel)@pucrs.br

***Abstract*—The Internet of Things (IoT) will connect billions of devices deployed around the world in a near future by embedding mobile network and processing power capabilities into a wide range of physical computing devices used in everyday life of many people. Recent studies concerning IoT have addressed not only the interoperability of devices, but also the context awareness feature which makes easy to discover, understand, and store relevant information related to IoT devices. This work aims to present a Context-Aware System called CONASYS, a system able to sense the environment and to provide contextualized services to the users, meeting the users needs without specific knowledge of the environment, and improving the Quality of Experience (QoE). We present in details the architecture of CONASYS, the technical issues related to the implementation of the system, and evaluation tests.**

I. Introduction

Internet of Things (IoT) is a novel computing paradigm that is rapidly gaining space in scenarios of modern communication technologies. The idea of the IoT is the pervasive presence of a variety of things or devices (e.g., RFID tags, sensors, smart phones, smart devices, etc), that are able to interact with each other and cooperate with their neighbors to reach common goals through unique addressing schemes and reliable communication media over the Internet [1]. Connect large numbers of devices directly to applications becomes infeasible and can bring scalability problems. In order to mitigate this inefficiency, middleware solutions have been introduced to address not only interoperability, but also many other functions as data management, security, and context-awareness.

The devices deployed around the world are generating a large amount of data, and, unless we can analyse, interpret, and understand these data, it will keep useless and with no meaning [2]. Context-aware computing has played an important role in tackling this challenge in previous paradigms, such as pervasive computing, which lead us to believe that it would continue to be successful in the IoT paradigm as well [3]. Context is considered any information that can be used to characterize the situation of an entity (e.g., a person, place, or computing device) that is relevant to the interaction between a user and an application, including the user and the application themselves. A context-aware system uses context to provide relevant information/services to the user, where relevancy depends on the user's task [3]. In this sense, an IoT environment requires a context-aware system in order to help the user in the most useful way.

In this work, we present the Context-Aware System (CONASYS), a system able to provide context-aware information in order to give semantic meaning (context) to entities and their data in IoT environments. The main intention is to avoid the manual user intervention in the interpretation of the data and also facilitates the systems/entities interactions.

The remainder of this paper is organized as follows: Section II presents some related work. Section III presents the details of our system. Section IV presents the evaluation and discussion. Finally, Section V presents the conclusions.

II. Related Work

Some systems provide context-aware functions to IoT environments. This section presents some examples of these systems and a brief review about their context-aware features.

Hydra [4] is an IoT middleware that comprises a Context Aware Framework that is responsible for connecting and retrieving data from sensors, context management and context interpretation. Feel@Home [5] is a context management framework that supports interaction between different domains. C-Cast [6] is a middleware that integrates WSN into context-aware systems. CA4IOT [7] is an architecture to help users by automating the task of selecting the sensors according to the problems/tasks at hand.

A set of methods is mandatory in order to obtain the context of an entity. Perera et al. [3] proposed a context lifecycle and explained how acquisition, modelling, reasoning, and distribution should occur.

Hydra, Feel@Home, C-Cast, and CA4IOT perform the acquisition process collecting information directly from the sensors. In CONASYS, PUSH and PULL methods are used in order to meet the heterogeneity of the devices. For context modeling, Hydra uses three techniques: Key-Value, ontology-based, and object oriented. C-Cast modeling through Markup Schemes technique. Feel@Home uses ontology-based technique. In CONASYS, both Key-Value and Markup Scheme techniques are used. These techniques are simple to use and facilitate the organization and storage of the context.

C-Cast, and Hydra work with rules for context reasoning. Hydra also makes use of ontology-based technique. CA4IOT uses semantic and statistical reasoning techniques. Feel@Home uses only ontology-based technique. Rules supplies the CONASYS necessities as it has a good cost-benefit compared to other technologies. For context distribution there

978-1-5090-6196-9/17/$31.00 ©2017 IEEE

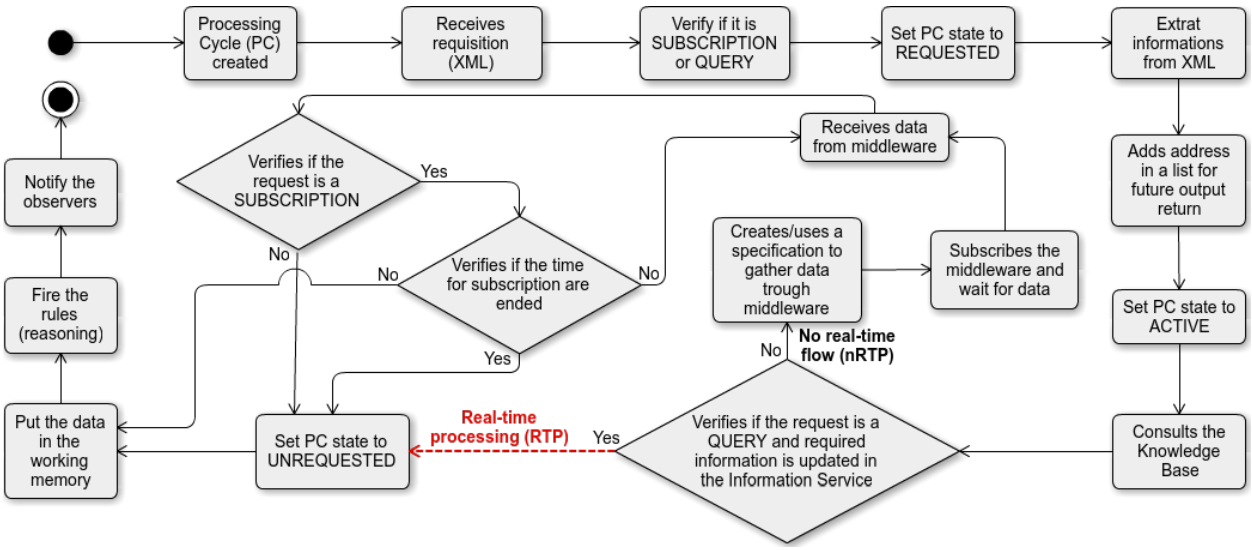

Fig. 1. Phases of the Processing Cycle (PC).

are two popular techniques: query and publish/subscribe. C-Cast, Feel@Home, CA4IOT, and CONASYS make the distribution through these two techniques. Hydra uses the query technique.

This study was made in order to elucidate how systems generate context. Moreover, only few systems provided details of their architecture and none of the studied systems have addressed real-time processing of the context, which has been considered a gap in the area [3].

III. CONASYS: Context-Aware System

The Context-Aware System (CONASYS) aims to provide to user/application a set of services of contextualized information, both on-line (i.e., through newest contextualized data) and off-line (i.e., through historical contextualized data). A user can request the services without knowing exactly which devices will be used in the process. CONASYS interconnects with COMPaaS (Cooperative Middleware Platform as a Service) to have access to the infrastructure of devices. COMPaaS is an IoT middleware [8] that lacks context-aware features in its architecture. However, we can use any middleware able to provide its devices as services. CONASYS can deal with any domain (e.g., smart home, smart office, healthcare, etc), since that each domain must have a specific set of rules registered in CONASYS.

A. Architectural Overview

An API was developed to allow users to interact with CONASYS. In addition to the API, CONASYS provides an architecture composed of three main layers: Communication Layer, Storage Layer and Processing Layer. The process of receiving and interpreting the user's request is made by the Communication Layer, that is also responsible for send results to users. The Storage Layer is responsible for storing content of all modules present in CONASYS. The Processing Layer is responsible for the context reasoning process. CONASYS uses Drools[1] rules for reasoning. As this work is focused in technical overview and platform evaluation, more architectural details can be seen in our previous work [9].

B. Technical Overview

CONASYS must recognize the environment that it is inserted. CONASYS is strictly connected to an IoT middleware. With a *thread* that receives all the updates of events that occur in the middleware, it is possible to sense (i.e., a kind of understanding) the environment. Every time that a device connects/disconnects/updates, it sends an XML file to the middleware. CONASYS gets this file, understands it, and stores the information. The key information acquired from the XML is the individual URI, that is the device identity.

Another important CONASYS background step is the rules interpretation. The rules are directly linked to the context reasoning process. The interpretation function has directives to decompose the rule in parts and store it. This process happens in order to provide the available services to user, since that every rule is linked to one or more services. The CONASYS set of rules must be defined and inserted in deployment phase. Rules is the simplest and most straightforward method of reasoning and are usually structure in an IF-THEN-ELSE format.

CONASYS implements the "Observer" pattern, so beyond the *query*, the user can also uses CONASYS by *subscription*. CONASYS receives the user request, by a specific communication API (through SOAP web service), understands it and creates the Processing Cycle (PC) (see Figure 1). First, PC verifies the request type (*query* or *subscription*), and extracts the services name from received request. PC also adds the address of the user in a list for future notifications. With the possession of the services name, PC consults the CONASYS

[1]http://www.drools.org/

```
rule "City_Pollution"
when
 obj1 : Output( service_device == "Air_152")
 obj2 : Output( service_device == "Ground_014")
 obj3 : Output( service_device == "Water_S01")
then
 String val_str1 = (String) obj1.getValue();
 boolean val_bool2 = (boolean) obj2.getValue();
 String val_str3 = (String) obj3.getValue();

 double value = Double.parseDouble(val_str1);
 if(value <=100 && value>=0) {
  if(value>55){
   obj1.setPriority(Priority.HIGHEST);
   } else {
      obj1.setPriority(Priority.NORMAL);
   }
  } else {
     obj1.setPriority(Priority.HIGHEST);
 }
 double value = Double.parseDouble(val_str3);
 if(value >=5.75) {
  obj3.setPriority(Priority.HIGHEST);
  } else {
     obj3.setPriority(Priority.NORMAL);
 }
 if (val_bool2 == true &&  obj1.getPriority()
       == Priority.HIGHEST) {
  if (obj3.getPriority() == Priority.HIGHEST) {
    obj1.setPriority(Priority.ALERT);
    obj2.setPriority(Priority.ALERT);
    obj3.setPriority(Priority.ALERT);
  }
 }
end
```

Fig. 2. Example of a smart city domain Drools rule.

Knowledge Base in order to be aware of which rule must be applied to each service.

At this point, PC query CONASYS database looking for events (i.e., device data). If there is a corresponding event in the database and the data is updated (i.e., depends of the device data generation time) and the user request is a *query*, the real-time processing (RTP) occurs and PC gets these data. With the RTP, PC goes to the context reasoning and distribution process. If there is no possibility of RTP, the no real-time flow (nRTP) continues. The RTP is like a shortcut of the PC. When nRTP occurs, the PC needs to get the data from middleware. When PC receives the middleware data, it verifies if user request type is *query*. If it is, PC goes to the context reasoning distribution process. If it is a *subscription*, PC verifies the time parameters of the request in order to end it or keep receiving middleware data.

After both RTP or nRTP, the PC starts the context reasoning and distribution process. The first thing of these processes is put the data in the working memory (reasoning process). After that, the rules are fired, which means that the data passes by the set of the CONASYS rules. The rules are in an IF-THEN-ELSE structure. In this sense, if any data reach the conditions of the rule, the actions of the rule are fired. This act can made data contextualization in many levels.

Context distribution process occurs when PC notifies its subscribers with these contextualized information. Moreover, there are different ways to send data to the user: Web Service, WebSocket and HTTP. In this way, CONASYS API can be used to change the method of communication. The user receives data through its *update* method and can interpret and use these data in countless ways.

TABLE I
EXECUTION TIME (MS) AND STANDARD DEVIATION FOR EACH RESULT OF MODELLING THE KNOWLEDGE BASE TEST SCENARIO.

Rules/Conditions	5	50	100
10	133,3 (8,8)	135,2 (6,6)	138,6 (7,3)
50	583,6 (19,1)	598,9 (20,8)	616,7 (21,5)
100	1164,9 (28,0)	1185,9 (29,3)	1197,1 (26,5)
200	2419,2 (49,4)	2426,2 (49,9)	2624,8 (51,9)
500	6269,1 (95,3)	6348,2 (107,7)	6520,5 (97,1)

Legend: *Execution time (standard deviation).*

IV. EVALUATION AND DISCUSSION

The goal of the tests is to evaluate CONASYS in terms of performance. All the tests were performed in three computers. First hosting CONASYS, second hosting the middleware and devices, and third hosting the application/user. All computers are equal with Ubuntu 14.04 LTS (64-bit), Intel Core i5-3230M 2.60GHz and 8GB RAM. The use of three computers in this evaluation is feasible since we are analyzing only CONASYS performance. The CONASYS was developed with Java programming language and PostgreSQL database.

A. Modelling the Knowledge Base

The main objective of this test is to measure the time taken by CONASYS to modelling a set of rules. The modelling of the Knowledge Base happens in two steps: (i) cutting each rule in parts (i.e., interpretation) and (ii) store it in CONASYS database. We create a synthetic Drools rule file with specific domain rules. For the tests, we modified the file in order to increment the number of rules. Moreover, the conditions of each rule also increased. These conditions are the field *when* of a rule. For example, the rule of Figure 2 only acts on 3 devices as we can see in the *when* field. In the most extreme tested scenario we have a Drools file with 500 rules and each rule with 100 conditions (i.e., *when* field). With this scenario we can reach, in an ideal setting, a total of 50000 (fifty thousand) devices (i.e., 500 rules · 100 conditions).

All the times collected were an average of 10 executions. The scenarios vary in two terms: number of rules in the file and number of conditions in each rule of the file. The tests results can be seen in the Table I. The first column of the table shows the number of rules in the file for each test. The first row of the table shows the number of conditions on each rule of the file.

Considering the test scenario with 100 rules in the file, the time taken by the system to interpret, model and store the rules was 1164,9 ms for 5 conditions in each rule. This scenario covers a total of 500 (five hundred) devices. If we increase the conditions of this scenario to 100, the scenario will cover a total of 10000 (ten thousand) devices. This conditions raising represents a 20 times (1900%) increase in the number of devices (five hundred to ten thousand), and the time taken

TABLE II
EXECUTION TIME (MS) AND STANDARD DEVIATION OF THE PROCESSING CYCLE (PC) WITH REAL-TIME (RTP) AND NO REAL-TIME FLOW (NRTP).

PC	10	20	50	70
RT	343,7(19,7)	504,9(27,2)	1243,3(39,1)	2232,7(58,2)
nRT	3442,1(75,8)	4531,8(83,4)	7560,9(102,1)	10605,4(131,8)

Legend: *Execution time (standard deviation).*

for the function process it suffers a increase of less than 3% (1164,9 ms to 1197,1 ms). For the critic tested scenario the time taking by the function was 6520,5 ms. Bearing in mind that this function only was triggered with the CONASYS deploy, it is an acceptable time. In most of the results, the standard deviation (SD) follows a stable line. This happens because independent of the rule file size, the process to model it stills with the same complex procedures (e.g., database access), while what changes is the simplest procedures (e.g., code interpretation) that do not have a big effect in the results.

B. Processing Cycle

In this test scenario, we collected the time taken by all the steps of the PC (see Figure 1), from start (i.e., receives the request interpreted) to end (i.e., contextualize and notify observers). We tested four case scenarios with different numbers of users requests fired simultaneously: from 10 to 70. The time was collected in two executions of each case scenario: (i) when the real-time processing occurs (RT) (i.e., device information updated in the database) and (ii) when no real-time flow occurs (nRT). We made ten executions of each test scenario in order to have an average execution time.

The first column of Table II shows the PC type that was performed: RT or nRT. The first row shows how many users requests were fired simultaneously, creating multiple PCs. As can be seen, the execution time of RT and nRT are very different. In the first test scenario, with 10 parallel executions and RT the obtained time was 343,7 ms, that is almost an instantaneous function. For the others tests, the time taken by RT increases linearly. When nRT occurs, is mandatory to have a connection with the middleware, which increases the time (i.e., communication delay). In the RT, standard deviation (SD) followed a pattern staying near to the recommended by [10] (i.e., the mean being approximately the square of the SD). For the nRT, the SD results do not followed this pattern. This happens because in the RT, CONASYS consults its own database. On the other hand, communication delay on a middleware query occurs in the nRT.

According to [11], real-time processing solutions are focused on processing faster than traditional methods, which allows stream data processing. In CONASYS, the RT shows up as a better alternative than nRT, since it meets the IoT needs providing the user response with a very small delay time. The RT acts as a shortcut, reduces the processing effort of the cycle and eliminates the need of a middleware connection, what could add a communication delay caused by the network.

A context-aware system that provides real-time information services and having a well-defined structure that is able to handle with different context situations is not yet defined and is necessary in IoT environments. IoT mobile ecosystems are in constant change, so the feature of providing contextualized information services in a real-time way becomes an important characteristic of our system. In addition, as the IoT can be applied to different situations, the possibility of working with multiple domains is also a strong characteristic of CONASYS. By having this features, CONASYS stands out comparing with other systems.

V. CONCLUSIONS AND FUTURE WORK

With context, we can give a semantic meaning to the "Things" data. A context-aware system uses context to provide services to the users (i.e., requester). We presented CONASYS, a system that give the context-aware capability to IoT middleware solutions and enables to build a sensing-as-a-service platform. Tests were performed in order to validate the CONASYS capabilities. We also show that CONASYS standing out by the real-time processing feature. In the future, we plan to make some improvements in order to refine the Quality of Experience (QoE) by enabling the rules update in runtime. In this sense, the user will be able to change the rules scope dynamically.

ACKNOWLEDGMENT

Our thanks to CAPES/CNPq for the funding within the scope of the PNPD project.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787 – 2805, 2010.

[2] A. B. Zaslavsky, C. Perera, and D. Georgakopoulos, "Sensing as a service and big data," *CoRR*, vol. abs/1301.0159, 2013.

[3] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *Communications Surveys Tutorials, IEEE*, vol. 16, no. 1, pp. 414–454, First 2014.

[4] A. Badii, M. Crouch, and C. Lallah, "A context-awareness framework for intelligent networked embedded systems," in *Advances in Human-Oriented and Personalized Mechanisms, Technologies and Services (CENTRIC)*, Aug 2010, pp. 105–110.

[5] B. Guo, L. Sun, and D. Zhang, "The architecture design of a cross-domain context management system," in *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, March 2010, pp. 499–504.

[6] E. S. Reetz, R. Tonjes, and N. Baker, "Towards global smart spaces: Merge wireless sensor networks into context-aware systems," in *2010 5th IEEE International Symposium on Wireless Pervasive Computing (ISWPC)*. IEEE, 2010, pp. 337–342.

[7] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Ca4iot: Context awareness for internet of things," in *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*, Nov 2012, pp. 775–782.

[8] L. A. Amaral, R. a. T. Tiburski, E. de Matos, and F. Hessel, "Cooperative middleware platform as a service for internet of things applications," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, ser. SAC '15. New York, NY, USA: ACM, 2015, pp. 488–493.

[9] E. d. Matos, L. A. Amaral, R. Tiburski, W. Lunardi, F. Hessel, and S. Marczak, "Context-aware system for information services provision in the internet of things," in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, Sept 2015, pp. 1–4.

[10] A. Hald, "Statistical theory with engineering applications," in *Statistical theory with engineering applications*. John Wiley & Sons, 1952.

[11] O. Kwon, Y. S. Song, J. H. Kim, and K. J. Li, "Sconstream: A spatial context stream processing system," in *2010 International Conference on Computational Science and Its Applications (ICCSA)*, March 2010, pp. 165–170.