The Rhadamanthys Quality of Context Architecture

Hélio Carlos Brauner Filho[1], **Claudio Fernando Resin Geyer**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{hcbfilho,geyer}@inf.ufrgs.br

Abstract. *With the rising number of Internet of Things devices and context providers, the need for sorting and selecting them according to user requirements and to the quality of sensed information becomes prevalent. Thus, this paper presents the Rhadamanthys architecture, designed to fulfill this requirement using Quality of Context information for both ranking and selection. Individual Quality of Context criteria as defined in the literature are evaluated for each available provider and used in formulae to obtain a single score that enables ranking and allows selection.*

1. Introduction

Estimates on the number of *Internet of Things*(IoT) capable devices in 2015 indicate around 15.4 billion in existence, with an expected growth towards more than 30 billion devices in 2020 and further to the mark of 75 billion devices in 2025 [IHS Markit 2016].

Due to the huge number of IoT context providers, researchers have been studying and proposing new methods to allow searching and ranking such devices according to the quality of their sensed and generated data in order to allow the selection of the most appropriate providers that can solve a given task.

Some solutions that can be cited include *Cuida* [Nazário et al. 2015], *Yasar's* [Yasar et al. 2011] and *CASSARAM* [Perera et al. 2013], which have in common the use of *Quality of Context* (*QoC*) definitions to evaluate several sensor information criteria individually, as well as the use of a final aggregation function that could make single ranking possible.

While *Quality of Context* was first defined as "any information that describes the quality of information that is used as context information" and that it "refers to information and not to the process nor the hardware component that possibly provide the information" [Buchholz et al. 2003], the concept of *QoC* has evolved to include measurement of sensed data quality using numerical values at least since [Kim and Lee 2006].

Considering the requirements for *IoT* sensor selection and the *QoC* definitions studied, and the fact that current solutions are not flexible enough to allow new methods for *QoC* evaluation and aggregation to be used, the *Rhadamanthys* architecture was conceived. It encompasses the process of searching, ranking and selecting providers, and allows flexibility in terms of *QoC* criteria and of aggregation methods to be used. It also includes a *Graphical User Interface* (GUI) to aid users in the task of informing their needs in order for the system to determine the *QoC* value of providers.

This paper is divided into 5 sections. Section 2 presents related work. The Section 3 presents the architecture and describes its components. Section 4 presents the prototype and evaluation of the architecture. Finally, Section 5 concludes it.

2. Related Works

The works presented here were selected because they make use of *QoC* concepts in their models but do not limit themselves to basic *QoC* definitions. Two other aspects considered were:

1. the use of individual criteria (referred to as *dimensions*, *attributes or parameters* in works such as [Perera et al. 2013] and [Nazário et al. 2015]) for evaluating *Quality of Context*, as well as an aggregation function that results in a single *QoC* value that could enable simple ranking and selection operations;

2. the use of *QoC* criteria along with a user interface for the selection of context providers.

In [Yasar et al. 2011], some *QoC* criteria are used in order to evaluate if messages in *Vehicular Ad Hoc Networks* (VANETs) should be transmitted or dropped so as to reduce network traffic and free resources. Definitions on how to evaluate four criteria, *Temporal Relevance*, *Completeness of Information*, *Priority of Information* and *Spatial Relevance*, are given, and a definition is given stating that each of those criteria might have different weights. The final *QoC* single value is obtained by calculating the weighted average of the criteria.

In *Trust and Privacy Management Support For Context-Aware Service Platforms* [Neisse 2012], *QoC* criteria are identified as *QoC attributes*, and are defined based on an international metrology standard found in [Vim 2004]. The *QoC* model defined in that work allows *QoC* values, such as *precision* of context information values and *freshness* of timestamps, to be obtained implicitly. The defined attributes help users decide if context providers are trustworthy concerning applications such as GPS positioning and temperature measurements.

The definition of *trustworthiness* found in it differs from what is shown in [Buchholz et al. 2003] and [Sheikh et al. 2008]. The latter two consider *trustworthiness* to be a *QoC* attribute, but [Neisse 2012] considers it to be "a property of the context provider from the context consumer's point of view, and is not a quality concept related to the context information itself" [Neisse et al. 2008]. Thus, *trustworthiness* is treated in the *Trust Management Model* and defined there instead of being defined in the *QoC Model*.

In *Context-aware sensor search, selection and ranking model for Internet of Things middleware* (*CASSARAM*) [Perera et al. 2013], the authors present a GUI based approach to complement their *Context Awareness for Internet of Things* (*CA4IOT*) [Perera et al. 2012] architecture. It allows priority assignment for several criteria to allow the selection of sensors according to user needs. By default, it has 30 criteria, called *context attributes*, available for users to select and assign priorities to using sliders, with possible values ranging from 0, *less priority*, to 1, *high priority*. The interface can be extended, allowing for more sliders to be added if new criteria are required by users to be considered.

After obtaining user priorities, weights are assigned to each of the criteria comparatively. Then, all of the values are aggregated into a single value using an index based on Euclidean distance calculation called *Comparative Priority-based Weighted Index* (*CPWI*). This index requires the user to inform an ideal sensor that is positioned in a

multidimensional space, with each space representing a criterion and ranging from *0* to *1*. Sensors with values closest to the ideal sensor will be selected. If no values are informed by the user, the ideal sensor has all values defaulted to *1*. The values for each sensor are then multiplied by the comparative weights resulting in a single *CPWI* value.

In *Cuida: um modelo de conhecimento de qualidade de contexto aplicado aos ambientes ubíquos internos em domicílios assistidos* [Nazário et al. 2015], an extensive study is made on *QoC* attributes in order to develop a knowledge model for *QoC* to be applied in a ubiquitous assisted household environment. The study identified a lack of standard nomenclature, standard definitions and standard quantification methods for *QoC* parameters. This resulted in the creation of an ontology model to represent *QoC*.

In order to evaluate the model, a context architecture is proposed, and within this architecture, two modules are dedicated to *QoC*. The first one is responsible for quantifying *QoC* parameters between *0* and *1*, using rules that were defined in [Nazário et al. 2014]. It is highlighted that the aggregation method, while using the same weighted average as [Yasar et al. 2011], differs from it in the way *significance* is treated as defined by [Manzoor et al. 2008]. While [Yasar et al. 2011] uses *significance* as one of the parameters in the weighted average, calling it *priority*, the approach in [Nazário et al. 2014] defines that this parameter should not lower the *QoC* value, but only be used to give different priorities when evaluating the information.

The second one is responsible for evaluating the QoC values that were obtained by the first QoC module, detecting possible problems with sensors and communication networks, or possible risks to the patient that is being monitored by the environment.

All of the related works presented *QoC* being used in some bigger context than just *QoC* definitions, showing that there are valid applications for *QoC* definitions, while a few others showed that having a user interface and quantifying *QoC* values between *0* and *1* is a trend and that those two aspects can make evaluation easier.

3. Rhadamanthys Architecture: Concepts and Model

The *Rhadamanthys Architecture* was designed to allow search, ranking and selection of context providers in a fashion similar to that of the *CASSARAM* model. Its main differences are the possibility of using different aggregation methods, the use of *QoC* profiles to make assignments easier for the user to make, and allowing the identification of redundant criteria by the user community.

3.1. Overview

The *Rhadamanthys Architecture* comprises 8 modules:

1. A *Search* module to discover the available context providers;
2. A *Weight and Requirement Assignment* module that receives assignments made by the user;
3. A *QoC Criteria Evaluation* module that evaluates each *QoC* criterion according, for the default criteria, to methods defined in the literature;
4. An *Aggregator* module that receives individual *QoC* criteria values and respective weights and returns a single *QoC* aggregated value;

5. A *QoC Profiles* module, that saves user weight and requirement assignment definitions in order to make them available to other users who might have the same needs;
6. A *Ranking* module that allows the use of different ranking methods for the values generated by the *Aggregator*;
7. A *Selection* module that uses the number of context providers informed by the user to select them based on their ranking;
8. An *Interface* that allows the user to inform weights, desired criteria etc. and shows the resulting selection.

A diagram of the modules encompassed by *Rhadamanthys* with the interactions between them is presented in Figure 1. Each of module is better explained in the following subsections.

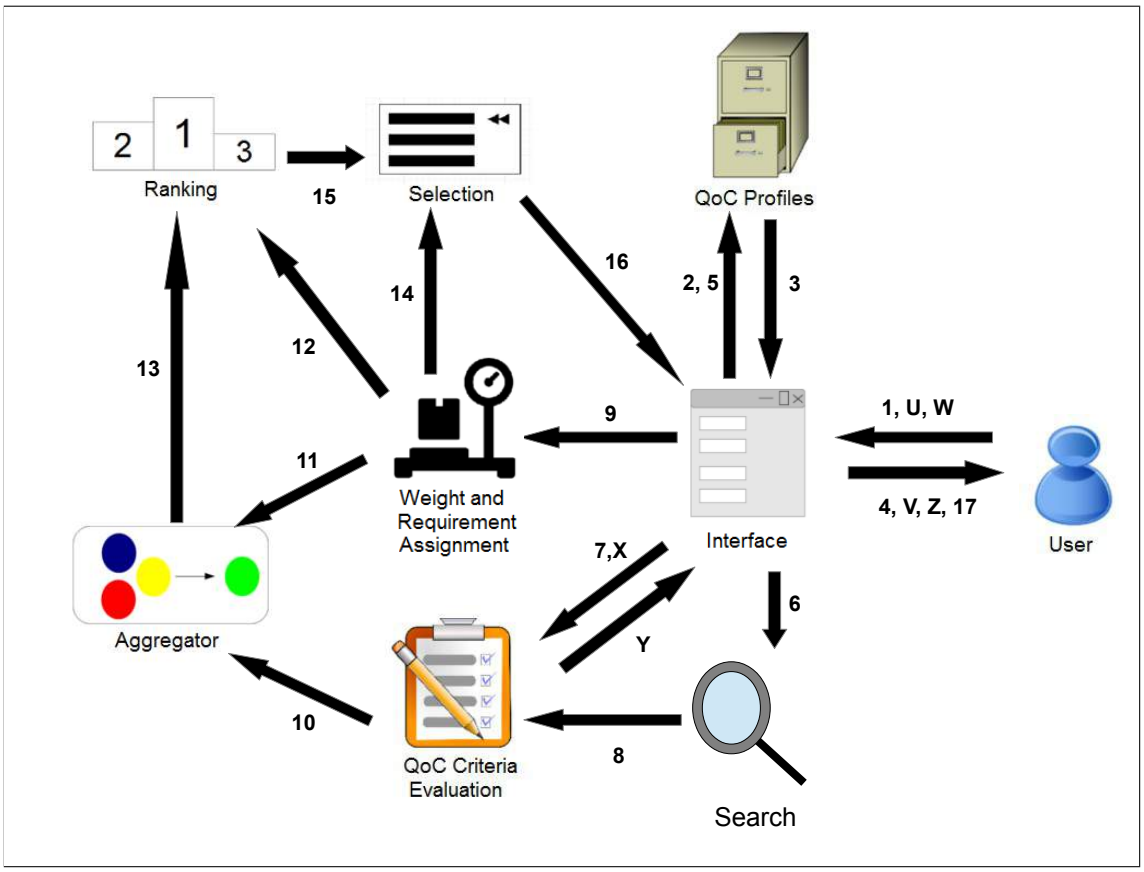

Figure 1. The Rhadamanthys Architecture with Modules and Interactions. The latter are described in the final subsection of Section 3

3.2. Search

The *Search* module finds available context providers to be tested in order to verify if they comply with user requirements. The module does so by keeping a list of all known providers and making requests for information when requested by the *Interface*.

3.3. Weight and Requirement Assignment

This module receives weights informed by the user, as well as other requirements, such as the maximum radius to consider context providers as valid candidates, the type of

provider that is required, the number of providers required, the aggregation method to be used and the desired ranking method. The selected ranking method is informed to the *Ranking* module, and the rest of the information is passed to the *Aggregator* module.

3.4. QoC Criteria Evaluation

The *QoC Criteria Evaluation* module is responsible for applying the formulae defined for each criterion to measure the corresponding *QoC* value, between *0* (*No Quality/Don't Care*) and *1*(*Best Quality*). These values are grouped according to the context provider that generated the data and sent to the *Aggregator* module.

3.5. Aggregator

The *Aggregator* module is responsible for receiving the values obtained by the *QoC Criteria Evaluation* module and the weights and other assignments from the *Weight and Requirements Assignment* module and apply the method chosen by the user to generate single *QoC* values for each context provider.

Since different aggregation methods might result in different execution times and different final results for each method, offering choices to the user allows the compromise between performance and efficiency according to need. The results are sent to the *Ranking* module.

3.6. QoC Profiles

This module is responsible for saving previous choices made by users according to an application description. To exemplify the advantage conferred by this module, one can imagine a user trying to describe his needs for an application that uses temperature measurement providers. It is possible that a previous user might have already described his needs for the same application. So, the current user might be able to use the same specifications the previous user defined by looking for the profile descriptions and finding *"Temperature measurement"* or a similar description and get the job done faster.

On the other hand, the current user might want to describe his own specification under the same description, or there might be redundant definitions for the same application with different descriptions.

In the first of the latter two cases, the *QoC Profiles* module must be able to save both descriptions and find balance between them in order to relay a single description to other users. This is ideally accomplished by an algorithm that is capable of generating a configuration with high representativeness towards the average, while discarding sets of user specifications that stray too much from it.

In the second case, definitions must be compared to see how close they are and be declared as candidates for merging. Merging can be either a manual process, when the scope of descriptions is small enough to be handled by humans, or an automated process otherwise.

3.7. Ranking

The *Ranking* module receives every value generated by the *Aggregator* module and ranks providers according to the ranking method informed by the *Weight and Requirement Assignment* module. These methods might vary according to how values are ordered (e.g.

ascending, *descending*) and to sorting algorithms used. The resulting ranking is then sent to the *Selection* module.

3.8. Selection

This module is selects the number of providers required by the user, and also for informs when the required amount of providers could not be met.

3.9. Interface

The *Interface* module receives all of the definitions and assignments the user makes and relay them to the other modules, and also receives information from these modules and show them to the user. Therefore, its role in the architecture is defined by how other modules and the user interact with it, and can be better understood by reading the *Interactions* subsection.

3.10. Interactions

The interactions represented by the arrows in Figure 1 are as described in this list:

1) The *User* interacts with the *Interface*
2) The *Interface* module requests current profiles to be shown to the *User*
3) The *QoC Profiles* module answers the requests made in *2*
4) The *Interface* informs the *User* which profiles are currently available
5) The *Interface* module sends values informed by the *User* to be saved by the *QoC Profiles* module
U) The *User* informs the *Interface* that a new *QoC* criterion will be added
V) The *Interface* requests a definition on how to evaluate this new criterion
W) The *User* informs, using some sort of language, the method used to evaluate this new criterion
X) The *Interface* signals the *QoC Criteria Evaluation* module that a new criterion and its respective method will be added
Y) The *QoC Criteria Evaluation* module signals that it has saved the information about the new criterion and its corresponding evaluation method
Z) The *Interface* informs the *User* that the operation started in *U* has been successfull
6) The *Interface* signals the *Search* module that the *User* has already defined his needs and is now expecting results, triggering the *Search* module to find context providers
7) The *Interface* informs the *QoC Criteria Evaluation* module which of the criteria must be evaluated according to the *User*
8) The *Search* module sends the information obtained from the context providers contacted to the *QoC Criteria Evaluation* module. The *QoC Criteria Evaluation* module applies methods to determine the value of individual *QoC* criteria in a range from *0* to *1*
9) The *Interface* transmits the weights and other assignments made by the *User* to the *Weight and Requirement Assignment* module
10) The *QoC Criteria Evaluation* module sends its results to the *Aggregator* module
11) *The Weight and Requirement Assignment* module sends its results to the *Aggregator* module

12) The *Weight and Requirement Assignment* module informs the *Ranking* module the ranking method to be used
13) The *Aggregator* module sends its results to the *Ranking* module
14) The *Weight and Requirement Assignment* module informs the *Selection* module how many context providers the *User* requires
15) The *Ranking* module sends the final ranking to the *Selection* module
16) The *Selection* module returns the final list of context providers that fulfill the *User* requirements to the *Interface*
17) The *Interface* returns the list to the *User*

4. Prototype and Evaluation

In order to evaluate the proposed model for the architecture, a prototype implementing its main functions was developed. The programming language used was *Java*, and the GUI was developed using the *Netbeans* IDE.

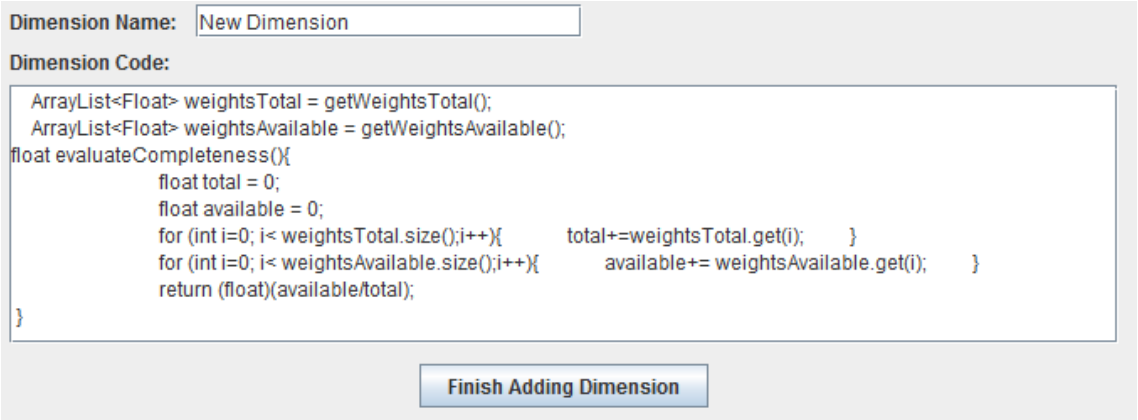

Figure 2. The "Add New Dimension" prompt window. It allows users to describe new methods for criteria through *Java* code

The *Interface*()Figure 3) has sliders ranging from *0* (*Not Important*) to *1* (*Most Important*), with big steps in order to save the user time fine tuning assignments. It also allows the addition of new criteria and their respective sliders and method codes through the *"Add New Dimension"* button, and the selection of ranking and aggregation methods (those are combined e.g. "Weighted Average - descending order"). Currently, the methods are described through *Java* code(Figure 2). The *Interface* also provides the option to select if context providers that are not of the specific type required by the user, but that can generate similar data (e.g using accelerometer information when the required provider type is *"seismometer"*) will be a part of the ranking.

The *QoC Criteria Evaluation* module was implemented using several of the definitions found in the literature, for parameters such as *Up-to-Dateness*, *Completeness* and *Battery Life*. Values vary from *0* to *1*, as modeled.

The *QoC Weight and Requirement Assignment* module also uses the *0* to *1* range variation, and does not use the comparative approach proposed in *CASSARAM*, but rather the defined weights directly.

The *QoC Profiles* implementation has used a simple method to verify its feasibility. An algorithm was implemented that picks the first definition related to an application

description as standard. The standard remains so until some other definition for the same application differs too much from it(1/3 of criteria with a difference of at least 0.2 between one user assignment and the other). When this happens before 10 definitions are made, the default action is to no longer show those definitions to users until it reaches that number of definitions, when the new definition is obtained through average. If it does not differ, the standard remains. While not ideal, this algorithm shows that, given the proper attention in order not to allow the problem of divergent definitions to cripple it, *QoC Profiles* can be a valuable aid for users to save time using predefined assignments. The current number of definitions required was defined as 10 to make tests easier.

The *Search* module currently reads a dummy file containing a list of descriptions for available providers and sends it to the *Interface*.

Based on the approaches shown in the literature, two aggregation methods were implemented for the *Aggregator* module: *Weighted Average* (Equation 1) and *Euclidean Distance Based* (Equation 2). *EDB* is based on the approach presented in [Perera et al. 2013], modified to solve a problem where the ranking was inverted. The equations for these two methods are described in the next subsection.

The implementation of these two showed that different methods can be used for aggregation. Also, tests using pseudo-random values showed that *Weighted Average* is more than two times faster and has better results than *EDB*, because the standard deviation for *WA* is smaller in at least 80 % of the cases, and in the cases where *EDB* has a better result the difference towards *WA* is smaller than a hunredth.

Also, three different ranking methods were implemented for the *Ranking* module, using *ascending*, *descending* and *random order*. This made it easier for different scenarios to be tested. The descending order makes it easier to detect failing context providers, and the random order can be used for statistical studies.

4.1. Equations

Equation 1 for the *Weighted Average* aggregated *QoC* value assignment is as follows:

$$WAvg = \frac{\sum_{i=0}^{n}(C_i * W_i)}{\sum_{i=0}^{n} W_i} \tag{1}$$

Where C_i is the quality value for criterion *i* obtained through the *QoC Criteria Evaluation* module and W_i is the assigned weight for criterion *i*.

Equation 2 for the *Euclidean Distance Based* method is described as:

$$EDB = 1 - \sqrt{\sum_{i=1}^{n}[W_i(U_i^d - S_i^\alpha)^2]} \tag{2}$$

Where W_i is the normalized weight assigned through the sliders for criterion *i*(that is, the ratio between the assigned weight and the sum of all assigned weights), U_i^d is the user-defined ideal value for criterion *i* and S_i^α is the value attributed to sensor α, (α being any sensor in the complete set of evaluated sensors) for criterion *i* by the *QoC Criteria Evaluation* module.

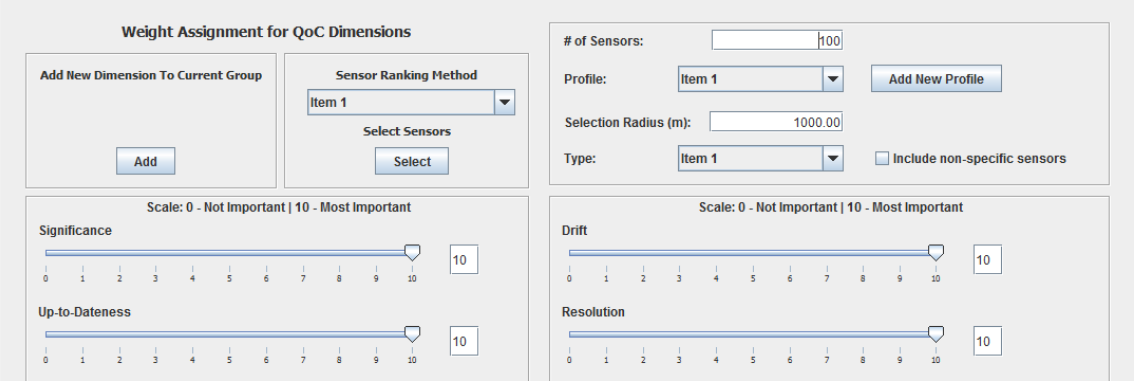

Figure 3. Interface implemented in Java. On the top left, the "Add New Dimension" button and the selector for ranking method and ordering. On the top right, options to select or add profile, define maximum radius, sensor type and required number. On the bottom, the sliders that enable weight assignment for criteria

5. Conclusion

The research made has shown that with the growing numbers of *IoT* capable devices, the need for methods that allow selection of devices amid those has increased.

Since the solutions shown in the related works that focus on *QoC* based selection did not present an architecture that allowed new criteria evaluation methods to be added, the *Rhadamanthys Architecture* was proposed, and its main aspects were detailed in this paper.

Advantages of the *Rhadamanthys Architecture* compared to other solutions include the *QoC Profiles* that enable users to inform their needs with ease, and the possibility of adding new criteria evaluation methods.

To prove the proposal feasible, a prototype implemented in *Java*, with a GUI developed with the *Netbeans* IDE was made. It confirmed that the architecture is capable of, within the described *QoC* value assignment model used, supplying different needs, allowing different configurations that can make it easier to detect the best or worst context providers and to make tests with random selections. It was also shown that the *QoC Profiles* might be an option to help users describe needs concerning context providers for applications.

This work has also shown that *EDB* is not an efficient method for aggregation, and the original algorithm in [Perera et al. 2013] results in inverted rankings.

For future works, evaluations to determine the overall performance of the system with large amounts of simulated providers are planned. Also, the implementation of a *QoC Criteria Evaluation* module to be installed on the *IoT* devices is going to be made, in order to verify if a performance gain on the server side where the main system is based is possible without sacrificing too much of the device battery, memory and processing power.

The interactions between the *Search* module and the providers should also be better studied. Other ways to describe methods for calculating criteria informed by the user will be studied. The evaluation will also involve at least one use case with opportunistic

networks. Interface improvements, new interactions and studies on security concerns are also planned.

References

Buchholz, T., Küpper, A., and Schiffers, M. (2003). Quality of context information: What it is and why we need it. In *Proceedings of the 10th HP OVUA Workshop, 2003*.

IHS Markit (2016). Complimentary Whitepaper: IoT Platforms - Enabling the Internet of Things. https://www.ihs.com/Info/0416/internet-of-things.html. [Online; accessed 19-March-2017], qtd. in https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2016/6101ec2b292d.

Kim, Y. and Lee, K. (2006). A quality measurement method of context information in ubiquitous environments. In *Hybrid Information Technology, 2006. ICHIT'06. International Conference on*, volume 2, pages 576–581. IEEE.

Manzoor, A., Truong, H.-L., and Dustdar, S. (2008). On the evaluation of quality of context. In *European Conference on Smart Sensing and Context*, pages 140–153. Springer.

Nazário, D. C., Dantas, M. A. R., and Todesco, J. L. (2014). Context management: toward assessing quality of context parameters in a ubiquitous ambient assisted living environment. *JISTEM-Journal of Information Systems and Technology Management*, 11(3):569–590.

Nazário, D. C. et al. (2015). Cuida: um modelo de conhecimento de qualidade de contexto aplicado aos ambientes ubíquos internos em domicílios assistidos.

Neisse, R. (2012). *Trust and privacy management support for context-aware service platforms*. Number 11-216. University of Twente, Centre for Telematics and Information Technology (CTIT).

Neisse, R., Wegdam, M., and Van Sinderen, M. (2008). Trustworthiness and quality of context information. In *Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for*, pages 1925–1931. IEEE.

Perera, C., Zaslavsky, A., Christen, P., Compton, M., and Georgakopoulos, D. (2013). Context-aware sensor search, selection and ranking model for internet of things middleware. In *Mobile Data Management (MDM), 2013 IEEE 14th International Conference on*, volume 1, pages 314–322. IEEE.

Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2012). Ca4iot: Context awareness for internet of things. In *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*, pages 775–782. IEEE.

Sheikh, K., Wegdam, M., and van Sinderen, M. (2008). Quality-of-context and its use for protecting privacy in context aware systems. *JSW*, 3(3):83–93.

Vim, I. (2004). International vocabulary of basic and general terms in metrology (vim). *International Organization*, 2004:09–14.

Yasar, A.-U.-H., Paridel, K., Preuveneers, D., and Berbers, Y. (2011). When efficiency matters: Towards quality of context-aware peers for adaptive communication in vanets. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 1006–1012. IEEE.