QoC-Aware Context Data Distribution in the Internet of Things

Pierrick Marie[1], Léon Lim[2], Atif Manzoor[2], Sophie Chabridon[2], Denis Conan[2], Thierry Desprats[1]
[1]IRIT UMR 5505 Université Paul SABATIER, 31062 TOULOUSE, France
<firstname>.<lastname>@irit.fr
[2]Institut Mines-Télécom/Télécom SudParis, CNRS UMR 5157 SAMOVAR, 91011 Évry, France
<firstname>.<lastname>@telecom-sudparis.eu

ABSTRACT
The Internet of Things (IoT) is a very dynamic and heterogeneous environment that generates plethora of sensor data, accessible to develop new smart pervasive applications. However, the substantial amount of effort required to collect and disseminate context data with sufficient quality prevents the context consumers to take advantage of the IoT to its full potential. Consequently, novel research efforts are required to design middleware solutions able to deliver relevant context data to consumer applications while hiding the complexity of data distribution in heterogeneous and large-scale environments. This paper presents the INCOME framework that enables context producers to express the level of Quality of Context (QoC) they are able to provide and context consumers to set thresholds on the QoC they expect in order to determine how to distribute context data. Our experiments show that context data can be annotated with QoC metadata and distributed from producers to consumers with a reasonable additional cost even on resource-constrained devices such as Raspberry Pi.

Categories and Subject Descriptors
C.2.4 [**Computer Systems Organisation**]: Computer Communication Networks—*Distributed Systems.*

General Terms
Design, Experimentation.

Keywords
IoT, Middleware, Distributed Event-Based Systems, Quality of Context

1. INTRODUCTION
By connecting any kind of things, the Internet of Things (IoT) paradigm allows the design of new applications in domains such as smart cities, smart homes, transportation and logistics. In addition to the communication standards proposed by the IETF (6LoWPAN, CoAP, etc.), the development of these applications will benefit from abstraction layers, and more specifically from concepts dealing with Quality of Context (QoC) aware context data distribution, that typically fall within the role of middleware. Since many of these smart things interact by pushing events, the publish/subscribe communication model [4] that is at the root of distributed event-based systems (DEBS) [10] is an important enabler: Their loosely-coupled interaction model decouples in space and time the things that produce events from applications that consume these events.

The IoT enables the collection of a large variety of context data, coming from local ambient sensors or remote sources such as cloud services. These context data can then be exploited by pervasive applications to detect the current situation of the users and provide them with the relevant services corresponding to their precise needs. However, context data are known to be imperfect and uncertain by nature [6]. Not underestimating the uncertainty of context data is crucial for enabling pervasive applications to behave properly. One way to limit this uncertainty is to manipulate additional knowledge associated to context data in the form of metadata. These metadata then represent the Quality of Context (QoC) [1] and allow to improve the operational value of the context [7] through various QoC criteria such as freshness, precision or correctness.

Even though event uncertainty has been recognized in DEBS, only a few works have made some proposals regarding this concern [8, 12]. This paper builds on our previous work on QoC modeling [9] and leverages the publish/subscribe model with QoC-based filtering. Our contribution consists in a content-based context data distribution framework where context producers express their QoC guarantees in advertisement filters and context consumers specify their QoC requirements in subscription filters. A matching mechanism specifically designed to consider context events with QoC metadata is then used at publication time.

The structure of the paper is the following. Section 2 presents a motivational scenario for our framework, which is then described in Section 3. Section 4 exposes the results of our evaluation of the cost of QoC-based filtering. Section 5 discusses some related works and Section 6 concludes the paper and expresses some perspectives.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first p age. C opyrights f or c omponents o f t his w ork o wned b y o thers than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
M4IoT'14 Workshop on Middleware for Context-Aware Applications in the IoT, December 08-12 2014, Bordeaux, France
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3234-7/14/12 ...$15.00.
http://dx.doi.org/10.1145/2676743.2676746

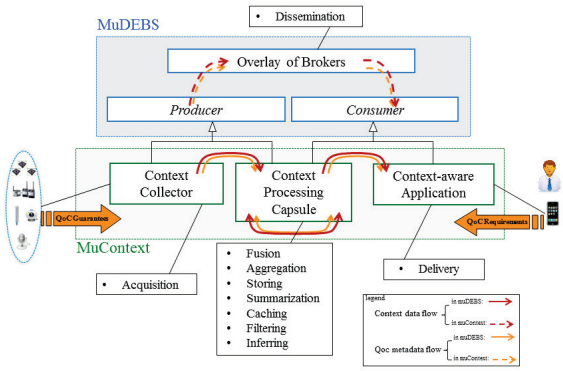

Figure 1: INCOME Logical Architecture

2. MOTIVATION

Bob arrives in Bordeaux to visit the city. He likes to walk or ride a bike. However, he is an asthma patient and very allergic to the polluted air. His host in Bordeaux advises him to install a pollution monitoring application to keep track of the air quality in the city streets. His host tells him that thanks to the INCOME framework, a lot of context-aware applications are now available in Bordeaux to inform and help the citizens about the current situation concerning different aspects of their life in the city. All those applications are using the INCOME framework for context collection and distribution.

Bob installs an application that receives INCOME context feeds about the air quality in the city streets. The INCOME framework uses QoC metrics to extract and aggregate the context information and generates high quality context feeds before sending notifications to concerned applications. This way applications get relevant information about the pollution level in the city streets. Bob configures the application according to his preferences and starts walking through the city streets following the application notifications. Bob visits Bordeaux historical places walking through the streets with good air quality and is very happy with his experience. The INCOME framework enables programmers to develop many interesting context-aware applications without investing any effort, time and money to collect context information and analyze the quality of the collected context information.

3. THE INCOME FRAMEWORK

The INCOME framework, as shown in Figure 1, consists of two major parts, muDEBS and muContext. muDEBS takes over the context data distribution meanwhile muContext deals with context management entities and their data models. muContext comprises context collectors, context processing capsules and context-aware applications. The context collector and the context capsule are inspired from the elements of the Context Toolkit [3] and of the Contextor [2] in that they are basic building blocks. In Section 2, context collectors, in collaboration with the city management services, collect raw sensor data (*e.g.* proportion of gases in the atmosphere) and metadata (*e.g.* location of sensors, precision of the measurements). Context processing capsules use sensor data to extract high level context information, such as the air quality at different locations in the city estimated as *good*, *moderate*, *unhealthy for sensitive groups*, *unhealthy*, *very unhealthy* and *hazardous*. They use the metadata to compute QoC indicators for that context information evaluated as *low*, *medium*, and *high*. Context processing capsules also work as context producers and publish the high quality context information along with QoC indicators to muDEBS, according to context contracts registered as advertisement filters. Finally, context information along with QoC indicators is provided to subscribed context consumers according to their requirements. This means that Bob can register a subscription filter indicating he wants to be notified when the air becomes *unhealthy for sensitive groups* and when this information is tagged with a *correctness* QoC indicator equals to *medium*.

The remainder of this section details the concerned part of the INCOME framework.

3.1 Context distribution

muDEBS is a framework offering content-based routing and is responsible for distributing context data. It is generic in the sense that it is data-model agnostic (data models are manipulated in muContext). The interface of muDEBS is the one of a distributed event-based system. Architecture elements called "producers" declare the kind of data they are willing to produce in "advertisements". Then, they publish these data in what are called "notifications" or "publications". Other architecture elements called "consumers" declare the information they want to receive and react to notifications delivered to them through "subscriptions".

muDEBS organises an overlay network of brokers that connect producers and consumers. Producers and consumers are collectively called clients. A client is connected to only one broker at a time; this broker is called the access broker. In content-based filtering, the filters managed by brokers in their routing tables evaluate predicates on the whole content of notifications [4]. In other words, there is no addition of routing metadata to notifications as done in topic-based filtering. muDEBS assumes that the data are semi-structured records serialised as XML schemas. The rationale for the choice of XML is its openness to allow approaches such as sensors as a service or ontology-based inference engines that often bring to play XML languages like RDF. It follows that we use XPath to navigate through XML data as standardized by the W3C.

An advertisement, which expresses the set of publications that a producer is allowed to publish, is kept local to the producer's access broker. Figure 2 shows the interaction between a producer and its access broker B_1. The filter F, which is uniquely identified by id, is advertised through the call to the operation advertise. The access broker registers the advertisement filter. Thereafter, when the producer publishes the notification n in the context of the filter identified by id, the access broker filters out n if n does not match F. In Figure 2, the advertisement filter is intuitively

drawn as an "input filter".

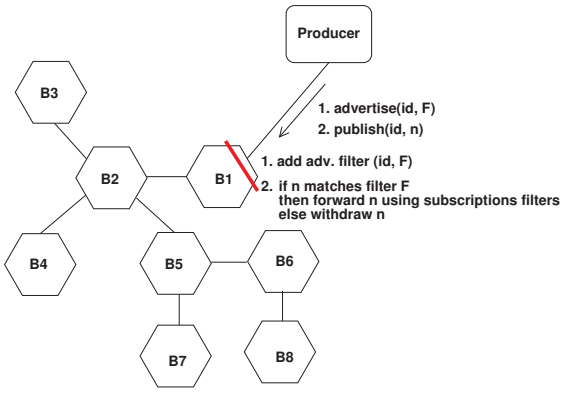

Figure 2: Effects of an advertisement on muDEBS brokers

In practise, an advertisement filter is a function, written in JavaScript, that evaluates XPath expressions and returns false when the notification does not match the filter, or returns true when it matches the filter.

A subscription expresses the set of publications that a consumer wants to consume. Figure 3 shows the interaction between the consumer and its access broker (and indirectly with the brokers of the overlay network) during a subscription. The subscription filter F' identified by id' is installed on every broker and leads to the formation of a spanning DAG (Directed Acyclic Graph) directed towards the consumer. The DAG is built so that producers can connect to any broker and so that their notifications will reach the consumers when they match F'. Therefore, when a producer publishes a notification n' associated to the advertisement filter id, the access broker of the producer applies the advertisement filter F. If n' matches F, then the access broker of the producer evaluates all the subscription filters it is aware of. When n' matches a subscription filter, let say F', the notification n' is forwarded towards the subscriber of F' via the access broker of the subscriber, which is notified of n'. In Figure 3, the subscription filter is intuitively drawn as an "output filter".

3.2 Software entities for QoC management

muContext involves three categories of software entities (see Figure 1): context collectors, context processing capsules, and context-aware applications. These categories implement a functional part of context management.

3.2.1 Context collectors and QoC management

A context collector is a software entity dealing with the acquisition of raw context data, *i.e.* that have not yet been processed or transformed. And it must be able to associate QoC metadata to raw context data. Since a context collector plays the role of an initial context producer, it manipulates QoC indicators relative to rather low level criteria. A low

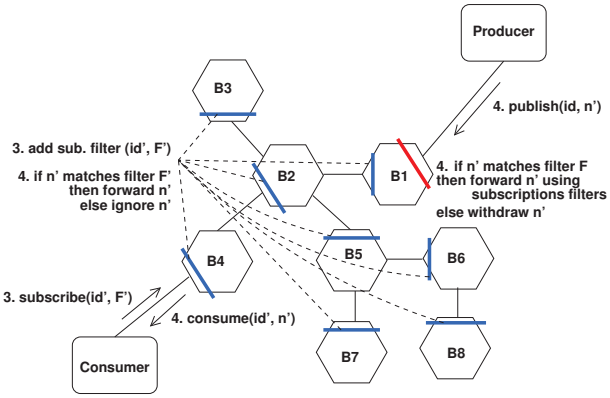

Figure 3: Effects of a subscription on muDEBS brokers

level criterion does not require computations or complex statistical analysis.

QoC specializes the general notion of Quality of Information (QoI) for context data. Freshness, precision and completeness are examples of QoC criteria. We have proposed in [9] the dedicated QoCIM (Quality of Context Information Model) meta-model, which offers a unified solution to model heterogeneous metadata about QoC. QoCIM includes constructs to: (i) associate QoC indicators to any context data. Each indicator is defined by one QoC criterion and can be valuated at runtime, in a dated way, by metric values; (ii) specify one or more definitions to characterize the metrics for the valuation of a criterion; (iii) define composite criteria in order to manipulate a single general QoC indicator instead of a set of individual indicators where each one corresponds to a simple QoC criterion. Consequently, QoCIM allows to exploit and manipulate QoC criteria in an expressive, computable and generic way.

3.2.2 Context processing capsules and QoC management

A context capsule is a functional element which performs the processing of context information into information of a higher level of abstraction made available to other capsules or context-aware applications. A context capsule is a context consuming and producing entity. The data produced are at a higher level of abstraction than the data consumed. It assumes the roles of intermediate consumers and producers in the chain of context information processing.

Several categories of context data manipulation can be operated by a capsule: aggregation, filtering, fusion, inference... The context management operations do not only perform a transformation of the context data flow. They also analyze what are the impacts on the management of QoC metadata during these manipulations which encompass more and less complex operations like: add / retrieve QoC indicators, update the value of an indicator, filter on the presence of an indicator or filter on the value of an indicator.

3.2.3 Contracting QoC

We take into account QoC requirements and guarantees through the notion of contract. Context producers express guarantees on the quality of the data they produce (symbolized by the arrow labelled G_{QoC} in Figure 1). Conversely, consumers express QoC requirements (symbolized by the arrow labelled R_{QoC}). These requirements and guarantees are then formalized in the form of context contracts on both the producer and consumer sides. Decoupled contracting is based on advertisement and subscription filters. An advertisement filter allows to express guarantees related to one or more QoC indicators, while a subscription filter specifies a requirement concerning some QoC indicators.

The specification of contracts and their translation into filters is of the responsibility of muCONTEXT while their implementation and evaluation are of the responsibility of muDEBS.

muDEBS relying on content-based filtering, filters are stored in the routing tables of the brokers and are evaluated as predicates on the notifications to be sent. These filters can relate to QoC indicators. muDEBS is agnostic of the data model, allowing to manipulate any model, for instance based on QoCIM filters.

4. EVALUATION OF THE COST OF QOC-BASED FILTERING

This section highlights the impact of QoC management on the performance of the INCOME framework. We first perform an experimental evaluation of the execution time of routing filters. We then present a quantitative analysis of an upper bound of the size of notification messages handled by the framework. Based on these results, a set of recommendations to write notifications and routing filters concludes this section.

4.1 Execution time of context and QoC-based routing filters

We relate in this section the experimental evaluation we have conducted on a prototype implementation of muDEBS and muCONTEXT for measuring the execution time of routing filters. For deciding whether a notification must be forwarded or not, the different constraints that compose a routing filter are evaluated. Constraints are XPath expressions. They control a subset of the notification and the content of a notification has to respect all of the constraints in order to be forwarded. The study consists in applying a routing filter to a single notification. The filters were configured to return *true* after a complete evaluation of the notification. The measurements of the execution time of routing filters presented in this section have been acquired with a machine equipped with an Intel i7 processor cadenced at 2.90 GHz and 4 GB of RAM and a Raspberry Pi machine[1], a low power machine that can be connected to the Internet of Things. To get realistic execution times, all the measurements correspond to the mean of 500 consecutive executions of routing filters.

[1] www.raspberrypi.org

4.1.1 Context-based filters

Before evaluating the performance of QoC-based routing filters, a first evaluation has been conducted to measure the execution time of context-based filters. The part of a notification message relative to context data is structured with three main elements: (i) a *context observable* is an abstraction that defines something to watch over (observe); (ii) a *context entity* is an element representing a physical or logical phenomenon (person, concept, etc.) to which context observables may be associated; (iii) a *context observation* is the state of an observable at a given time. An URI identifies the context entity and the context observable. For the rest of the study we consider context-based filters as routing filters composed with only one context-constraint that exclusively rely on these URI. Listing 1 presents an example of this kind of constraints. The constraint requires the pollution resource measured with the sensor number 45 placed in the Thiers avenue in Bordeaux. The execution time of this kind of filters is approximately 58 ms on the desktop machine and approximately 1045 ms on the Raspberry Pi.

4.1.2 QoC-based filters

After measuring the execution time of context-based routing filters, the second part of the study is focused on QoC-based routing filters. For this evaluation, two types of constraints relative to QoC metadata have been expressed. (i) A QoC-criterion constraint controls if the metadata of a notification include all the expected QoC criteria. It therefore evaluates the attribute *id* of the classes *QoCIndicator*, *QoCCriterion* and *QoCMetricDefinition* of the QoCIM meta-model [9]. (ii) Like the first type of constraints, a QoC-value constraint controls QoC criteria. It additionally controls the value of each QoC indicator by evaluating the attribute *value* of the class *QoCMetricValue*. Listing 1 presents an example of these two types of constraints. In the example, both constraints require the criterion 10 while the QoC-value constraint also requires a QoC metric value larger than 40.

Listing 1: Examples of constraints used for the study

```
// Context-based constraint
if( xpath.evaluate("//observable[uri='#pollution' and
 entity[uri='bordeaux://thiers_ave./sensors/45/']]",
 doc, XPathConstants.NODESET).length == 0) {
        return false; }

// QoC-criterion constraint
if(xpath.evaluate("//qocindicator[@id='10'
 and qoccriterion[@id='[10.1]']/
 qocmetricdefinition[@id='10.1']]", doc,
 XPathConstants.NODESET).length == 0) {
        return false; }

// QoC-value constraint
if( xpath.evaluate("//qocindicator[@id='10' and
 qoccriterion[@id='[10.1]']/
 qocmetricdefinition[@id='10.1']
 and qocmetricvalue[@value>='40']]", doc,
 XPathConstants.NODESET).length == 0) {
        return false; }
```

To compare the execution times for QoC-criterion and QoC-value constraints, we used two routing filters. The filter A contains one QoC-criterion constraint while the filter B contained one QoC-value constraint. For a notification with 256 QoC metadata, the biggest notification used in the study, the desktop machine spends respectively 538 and

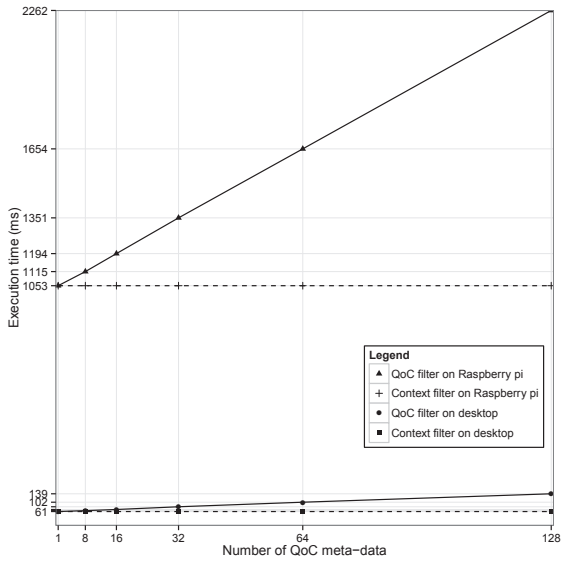

Figure 4: Execution time of QoC-based filters depending on the number of QoC metadata

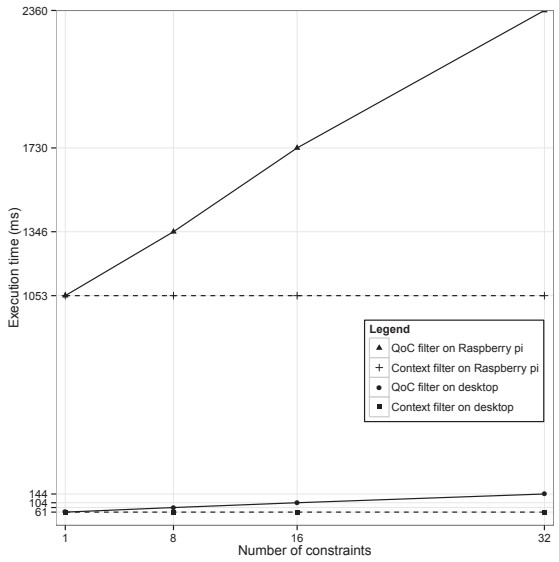

Figure 5: Execution time of QoC-based filters depending on the number of constraints

576 ms to execute the filters A and B. With the same configuration, the Raspberry Pi machine spends respectively 10480 and 11290 ms. There is not a significant difference between the execution time of the filters A and B. Therefore we consider for the rest of the study QoC-criterion and QoC-value constraints as equivalent and the measurements are made with QoC-value constraints.

Figures 4 and 5 compare the execution time of different routing filters. The dashed lines indicate the execution time of the context-based filters presented in Section 4.1.1. The Figure 4 presents the results of the experimentation where the notifications vary while the QoC-based filters do not change. For this experience, the filter is composed of only one QoC-value constraint and the notification contains one observation with 1 to 128 different QoC metadata. The slopes of the corresponding results are respectively 0.6 and 9.5 for the desktop and the Raspberry Pi machine. Conversely, another experimentation has been conducted where the notifications do not change but the QoC-based filters vary. Its results are displayed in Figure 5. In that case, the notification contains one observation with only one QoC criterion whereas the filter contains 1 to 32 different QoC-value constraints. For the desktop machine, the slope of the results is 2.7 and 42 for the Raspberry Pi machine.

For the same machine, the slopes of the graphs in Figure 5 are 4 times greater than the slopes of the graphs in Figure 4. That means the number of constraints within routing filters has more impact in terms of execution time than the number of QoC meta-data contained within notifications. Moreover, the figures indicate the execution time of the filter follows a linear curve and provides a solution to predict the execution time for higher number of QoC meta-data or constraints.

4.2 Estimation of the size of notifications

	Size of one observation	Size of one piece of QoC metadata
Empty XML document	426 char	444 char
Real XML document	$\simeq$ 600 char	$\simeq$ 800 char

Table 1: Size of observations and QoC metadata

As explained in Section 3.1, notifications exchanged between producers and consumers are serialized as XML documents. For this study, we used the first version of the framework that provides an unoptimized serialization method and produces uncompressed documents. Therefore, the XML documents are very verbose. For example, a QoC indicator is declared as follows: <qocindicator id="10" name="PrecisionQoCIndicator">...</qocindicator>. So, this study estimates the number of characters in the worst case, where the size of the notifications is the largest.

A notification contains at least one observation (the context data) and optional QoC indicators (composed of a value and a definition) based on the QoCIM meta-model [9]. Table 1 summarizes the size of XML documents following the context model of muContext. The size of an empty XML structure of an observation is close to the size of a QoC indicator, but with real data a QoC indicator needs more characters than an observation. Two reasons explain this difference. Firstly, the QoCIM meta-model contains more attributes than the context model used to represent an observation. Secondly, many attributes of QoCIM are based on String and include verbose information. Using Table 1, it is possible to estimate the size of a notification with the following formula:

$$size = 600 * nb\ observations + 800 * nb\ QoC\ metadata$$

A notification with one observation and one QoC indicator then contains 1400 characters. If the notification includes the 17 QoC indicators identified in the state of the art of [9],

its size becomes $600 + 17 * 800 = 14200$ characters. So, with an unoptimized serialization method, the size of notifications is significantly impacted by the number of QoC metadata.

4.3 Discussion

We consider a notification with 8 QoC meta-data as usual because it can cover many use cases like the scenario described in Section 2. In that case, Figure 4 indicates the execution time of a routing filter increases by 10% compared to a notification with only 1 QoC meta-data. In the same way, a routing filter with 8 constraints can express many requirement and cover various scenario. In that case, Figure 5 indicates the execution time of a routing filter increases by 30% compared to a filter with only 1 constraint. So, the impact of adding QoC management to routing filters is negligible as long as QoC-based filters do not require to evaluate more than 8 QoC-value constraints or notifications do not include more than 8 QoC metadata.

If the execution time of QoC-based routing filters is considered as acceptable when it corresponds to less than two context-based filters, they may contain up to 16 QoC-value constraints. Similarly, a notification that includes up to 64 QoC meta-data is analysed by a QoC-based filter with one QoC-value constraint in the same time taken by the evaluation of two context-based based filters. But in that case, the size of notifications becomes large and using compressed XML documents appears mandatory.

5. RELATED WORKS

A few works have started to consider the uncertainty of context data during the dissemination phase. [5] proposes a context data distribution infrastructure for query-based applications. Cache management strategies then rely on QoC for keeping only fresh data in the cache. We instead consider that future pervasive applications will benefit from event-driven thinking more than from a traditional request/response model. [8] considers quality-aware data stream management systems based on a relational model and with a probabilistic processing for the evaluation of quality of context. Even though such an approach is promising, it remains very sensible to the choice of the system parameters for the probabilistic processing. [11] describes a quality-aware publish/subscribe system for mobile sensor networks. It proposes to rely on location-based routing to deliver the subscriptions to the corresponding areas of interest. However, only event consumers may express their QoC expectations and no advertisement is performed on the side of event producers. We believe that a more powerful filtering can be obtained with content-based routing which benefits from both consumer requirements through subscriptions and producer guarantees through advertisements.

6. CONCLUSION AND PERSPECTIVES

This paper proposes to add QoC-based filtering in a DEBS infrastructure as a middleware solution for an efficient context data distribution in the IoT. We rely on a generic DEBS pattern and a generic QoC modeling approach which are both agnostic of the context model to address the heterogeneity of the IoT. The evaluation results on a first prototype implementation with no optimization show that the cost of QoC-based filters is reasonable.

One keypoint of our approach is that it was designed to enable privacy management in future work. The knowledge of the set of QoC indicators that are expected by consumers and offered by producers allows to finely tune privacy policies.

Acknowledgments. This work is part of the French National Research Agency (ANR) project INCOME[2] (ANR-11-INFR-009, 2012-2015). The authors thank all the members of the project that contributed directly or indirectly to this paper.

7. REFERENCES

[1] T. Buchholz, A. Kupper, and M. Schiffers. Quality of Context Information: What it is and why we Need it. In *10th Int. Workshop of the HP OpenView Univ. Association (HPOVUA)*, Geneva, July 2003.

[2] J. Coutaz and G. Rey. Foundations for a theory of contextors. In *Computer-Aided Design of User Interfaces III*, pages 13–33. Springer, 2002.

[3] A. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.

[4] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2), June 2003.

[5] M. Fanelli, L. Foschini, A. Corradi, and A. Boukerche. QoC-Based Context Data Caching for Disaster Area Scenarios. In *IEEE Int. Conf. on Communications, Kyoto, Japan, 5-9 June*, pages 1–5, 2011.

[6] K. Henricksen and J. Indulska. Modelling and using Imperfect Context Information. In *Proc. 1st PerCom Workshop CoMoRea*, pages 33–37. IEEE Computer Society, Mar. 2004.

[7] N. Hönle, U.-W. Käppeler, D. Nicklas, T. Schwarz, and M. Großmann. Benefits of Integrating Meta Data into a Context Model. In *3rd IEEE Conf. on Pervasive Computing and Communications Workshops*, pages 25–29, Kauai Island, HI, USA, Mar. 2005. IEEE Computer Society.

[8] C. Kuka and D. Nicklas. Quality matters: supporting quality-aware pervasive applications by probabilistic data stream management. In *The 8th ACM Int. Conf. on Distributed Event-Based Systems, DEBS, Mumbai, India, May 26-29*, pages 1–12, 2014.

[9] P. Marie, T. Desprats, S. Chabridon, and M. Sibilla. QoCIM : a Meta-model for Quality of Context. In Springer, editor, *CONTEXT'13: 8th Int.Interdisciplinary Conf. on Modeling and Using Context*, volume 8175, Oct. LNCS 2013.

[10] G. Mühl, L. Fiege, and P. Pietzuch. *Distributed Event-Based Systems*. Springer, 2006.

[11] E. C. Ngai and P. Gunningberg. Quality-of-information-aware data collection for mobile sensor networks. *Pervasive and Mobile Computing*, 11:203–215, 2014.

[12] S. Wasserkrug, A. Gal, O. Etzion, and Y. Turchin. Efficient Processing of Uncertain Events in Rule-Based Systems. *IEEE Trans. on Knowledge and Data Engineering*, 24(1):45–58, 2012.

[2] http://anr-income.fr