Enabling self-configuration of QoC-centric fog computing entities

Pierrick Marie, Thierry Desprats, Sophie Chabridon, Michelle Sibilla

Enabling self-configuration of QoC-centric fog computing entities

Pierrick MARIE*, Thierry DESPRATS*, Sophie CHABRIDON†, and Michelle SIBILLA*
*IRIT UMR 5505, Université Fédérale de Toulouse, 31062 Toulouse, France
Email: <Firstname>.<NAME>@irit.fr
†SAMOVAR, Télécom SudParis, CNRS, Université Paris-Saclay, 9 rue Charles Fourier, 91011 Évry Cedex, France
Email: <Firstname>.<NAME>@telecom-sudparis.eu

***Abstract*—The advent of the Internet of Things (IoT) enables the development of new applications and context-aware services. However, for applications requiring a real-time consciousness of their environmental conditions, some additional mechanism is necessary. The paradigm of fog (or edge) computing is a promising candidate to meet this requirement by supporting the deployment at the network edge of entities for pre-processing the data produced by the IoT. Thus, the acquisition, filtering and processing (aggregation, fusion...) of contextual data can be performed locally in real-time within software entities deployed on equipments of a fog. As context is central in the targeted applications, qualifying context information becomes essential. Meta-data may therefore be added including some quality criteria such as precision, freshness, completeness, for measuring the Quality of Context (QoC) information. QoC management must take place throughout the whole processing chain of context information, impacting the operations performed within the entities of the fog. Facing the potential physical limitations of the equipments at the network edge, this paper promotes declarative programming of processing entities able to qualify context information and self-reconfiguration. Thus, the parameters controlling the transformation and qualification operations may be adjusted based on the observation of resource usage. A prototype shows that the reconfiguration time does not exceed one second and remains within acceptable limits for the targeted applications. This solution offers an alternative to the principle of offloading code advocated by some works on Fog Computing.**

I. Introduction

The Internet of Things (IoT) brings the opportunity to spatially and temporally extend the informational scope of what constitutes the context of an entity. [1] defined context information as "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves." One solution to handle the context information is to use Context Managers (CM). A Context Manager is a kind of middleware that provides services to manage context information throughout its life cycle. As described in [2], the IoT will allow any connected object or sensor to regularly produce and share information about itself and its own immediate environment. Consequently, each piece of IoT-based information is a candidate for being processed by Context Managers with the objective to obtain an extended and enhanced contextual information. One major challenge in context computing is to deal with the distribution of Context Managers. The work presented in this article joins this approach to obtain IoT-based Distributed Context Managers (DCMs) such as the ones targeted by the INCOME project [3].

Thanks to wireless networking and mobility-supported devices, a new category of applications now becomes possible. Such applications ubiquitously provide users with services that are highly sensitive to any real-time variation of the context of the user. This context can cover the proximate environment of a user but can also include entities that are currently located in remote spaces. In both cases, data produced by local or distant IoT-connected sensors and smart objects constitute the raw material from which a context information can be computed. Starting from the acquisition of raw data, following by one or more transformation operations and ending by the distribution of the high-level context to the sensitive application, the whole context computing activity has to be achieved in an acceptable time limit.

The main objective of this work is to provide the developers of new real-time IoT-based context-sensitive applications with middleware programming facilities. An introduction of the Fog Computing and the QoC management is presented in Section II. Section III surveys recent solutions for context management and identifies relevant processing functions to easily manipulate QoC meta-data together with context information. Before presenting our implementation of programmable and self-reconfigurable Fog Computing entities in Section V, Section IV specifies functions for processing QoC and context information. Finally, Section VI concludes this work.

II. QoC-centric fog computing entities

Facing both the requirements of the real-time and IoT-based sensitive applications, and also the amount of data continuously produced by the IoT, many authors as [4] or [5] demonstrate the limitation of Cloud Computing in terms of latency, lack of mobility support, geo-distribution and location-awareness [4]. The paradigm of Fog Computing is proposed as complementary to Cloud Computing to address the requirements of applications that do not fit with Cloud Computing. According to Cisco [6], Fog Computing extends the Cloud Computing paradigm to the edge of

the network. From a context management point of view, software components that are responsible for transforming, in a time-constrained fashion, raw IoT-based data into context could find in the Fog layer a very suitable operational environment.

As described by [6], Fog Computing may be used to improve a traffic light system by deploying a Fog Computing node at each intersection. With the goals of preventing accidents and maintaining a steady flow traffic, the nodes measure the distance and speed of vehicles and detect the presence of pedestrians or cyclists. In real time the system analyses, reacts and changes the traffic lights following the incoming vehicles or sends an urgent alarm when a collision is anticipated within a few milliseconds. Another Fog Computing use case is described by [4] and concerns the augmented reality and real-time video analytics. The authors present Fog Computing as a solution to collect video stream or process object recognition for augmented reality applications deployed over resource limited devices like connected watches or glasses.

Context data may change very frequently and are inherently flawed since they come from devices, such as sensors, having inherent physical limitations or inaccuracy linked to their external environment. As context is central in the targeted applications, it is of paramount importance to limit the impact of its imperfections. One solution consists in adding quality meta-data to context information and then in considering the quality of context information (QoC) together with the context itself. In this article we define QoC "as the set of parameters useful to express properties and quality requirements on context data" [7]. Therefore, QoC must be considered in an end-to-end manner all along the lifecycle of context information during the acquisition, transformation and distribution steps of context management.

When considering real-time QoC-centric applications, a static QoC management is not suitable as it will not allow to adapt to the changing environment and to the available computing resources. Therefore, this paper investigates how to (re)configure context and QoC processing software components to be deployed at the network edge close to the context producer and consumer entities. Especially, this work is focused on the self-adaptation of the transformation processes executed by these components. In this article, we use the term "capsule" to designate the context and QoC processing software components deployed over Fog Computing nodes. A capsule is a functional element of the context and QoC management service that is packaged as a unit of deployment. Capsules are deployed as close as possible to context information sources with the purpose of clean, analyse and transform the information coming the IoT and and supply at runtime and on demand end-users applications.

The next two Sections present our work on specifying both context and QoC processing functions including the settings that can govern their behaviour. Based on this specification, Section V details a declarative programming approach for helping developers to implement self-reconfigurable capsules.

III. Elicitation of context processing functions

This Section analyses four context managers [8]–[11] and two states of the art, one concerning context data distribution [12] and one concerning context-aware computing [13]. The name of the functions used by the authors is presented in this Section. The purpose is to identify the most used context processing functions to characterise their operational behaviour.

A. Nurmi et Floréen (2004) [8]

Independently of any context manager implementation, the authors propose four approaches for context-reasoning tasks. In this paper, we consider the first approach named "low-level approach". It contains three functions named *pre-processing*, *fusion* and *context-inference*. Pre-processing aims to clean data, handle missing information and identify relevant information. Fusion combines and integrates information coming from different sources. The function must not produce outliers that are rejected in further analysis. Finally, context inference consists in identifying new context information and mapping low level contexts to higher level contexts.

B. Sehic et al. (2012) [9]

The authors propose a context manager named "Origins Toolkit". It carries out four functions: *filtering*, *aggregation*, *composition* and *inference*. Like [8], the authors use a function named *inference*. The filtering function, suggested by [9], performs some of the operations supported by the pre-processing function of [8]. The authors summarize the behaviour of their function in Figure 1. For each function, the Figure details the type of information consumed by the function and the type of information that it produces. In the Figure, every geometric shape represents a type of information. "Or" designates a source of context information and "CA" a context application.

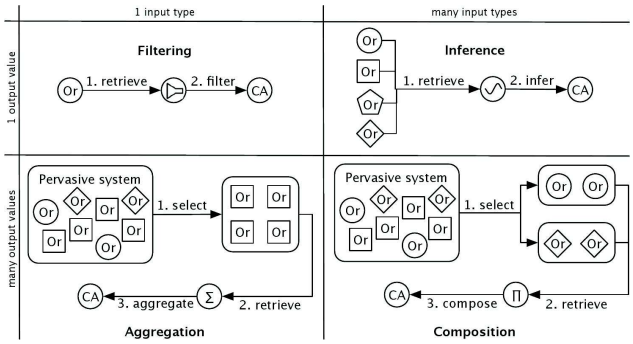

Fig. 1. Sehic et al. (2012): "Processing Operations" [9]

C. Filho et Agoulmine (2011) [10]

Different modules compose the architecture of the context manager proposed by the authors. Three of them handle context processing functions. The context collector module *aggregates* and *stores* information coming from context

sources. *Inference*, *fusion* and *derivation* are placed into the context reasoner module. Finally, the context obfuscator module deals with the *obfuscation* and *anonymisation* functions. Compared to [8] and [9], the authors propose new functions like storage, derivation, obfuscation and anonymisation. The behaviour of the storage function is easily interpretable, it allows the context manager and context application to get historical information. But no more detail is provided to clearly distinguish the aggregation and derivation functions. In the same way, obfuscation and anonymisation, related to privacy management, can be implemented with many algorithms resulting in a different behaviour. But this issue is not covered by [14].

D. Manzoor et al. (2014) [11]

Another context manager based on components is proposed by [11]. It performs many tasks such as *fusion*, *extraction*, *filtering*, *aggregation*, *composition* and *storage*. Extraction is a new function name that appears in this list. But there is no detail in [11] to distinguish the behaviour of these functions. Moreover, the difference between the derivation function proposed by [10] and the extraction function proposed by [11] is not clearly established.

E. Bellavista et al. (2012) [12]

[12] provides a state of the art concerning the context data distribution for mobile ubiquitous systems. A part of the taxonomy used by the authors to classify the studied solution concerns the context processing methods. The authors identify and describe four categories of functions: *context data history*, *context data aggregation*, *context data filtering* and *context data security*. All of the functions listed in the previous Sections could be classified into one of these categories. Context data history provides solutions to store relevant past events and retrieve the history of a particular value. Context data aggregation consists in merging and combining context data. Context data filtering increases the scalability of the system by controlling and reducing the amount of transmitted data. Finally, context data security includes the mechanisms to grant the privacy, integrity and availability of the data.

F. Perera et al. (2014) [13]

Almost fifteen context managers are compared in [13]. The authors use different categories to classify the context managers as their context acquisition method, the models or the ontology used to represent context information, or the reasoning functions used to produce new context informations. Concerning the reasoning functions, the authors use three types of function: *pre-processing*, *fusion* and *inference*. The same names are also used by [8] to identify their functions. Pre-processing functions are used to clean data by removing outliers, filling missing information or validating context. Fusion combines different pieces of information to generate a new one that is more accurate and complete and could not be achieved with a single context source. Finally, inference functions produce high level context information from low level information.

G. Summary

Figure 1 proposed by [9] provides the most complete description of processing functions. It includes the number of input and output pieces of information, one or many, and the type of information accepted and then produced by the function. In the other works, only a textual description is sometimes proposed.

Function	[8]	[9]	[10]	[11]	[12]	[13]	**Occurrences**
Aggregation		✓	✓	✓	✓		**4**
Fusion	✓		✓	✓		✓	**4**
Inference	✓	✓	✓			✓	**4**
Storage			✓	✓	✓		**3**
Filtering		✓		✓	✓		**3**
Anonymisation			✓		✓		2
Composition		✓		✓			2
Obfuscation			✓		✓		2
Pre-processing	✓					✓	2
Derivation			✓				1
Extraction				✓			1

TABLE I
OCCURRENCE OF THE NAME OF THE FUNCTIONS IN THE LITERATURE

Table I summarizes the functions identified in this Section. The Table classifies the functions according to their number of occurrences in the literature. The five most used functions are *aggregation*, *fusion*, *inference*, *storage* and *filtering*.

The purpose of Section IV is then to specify the behaviour of the context management functions identified in Table I and defining the links between context and QoC management.

IV. Definition of the QoC-aware context processing functions

With the objective to define context processing functions and their links with the QoC management, the first part of this Section is dedicated to formalize six functions used to manipulate QoC meta-data. These functions provide a solution to easily manipulate QoC meta-data associated to context information. Then, the second part of this Section defines the behaviour and the parameters of context processing functions. Because QoC management has to be handled as well as context processing, Section IV-B also highlights the dependencies between the context processing functions and the functions used to manipulate QoC meta-data.

A. QoC management functions

The model of QoC meta-data manipulated by the functions is the QoCIM meta-model [15]. Every QoC meta-data handled by the functions is an instance of a QoC indicator modelled with QoCIM. To clarify the definitions of the functions, Listings 1, 2 and 3 respectively formalize three types of data: `QoCMetaData` based on QoCIM, `ContextInformation` with basic information concerning the context and `Message` that represents a `ContextInformation` associated to a list, possibly empty, of `QoCMetaData`.

```
data QoCMetaData = { QoCIndicator.id::Int,
 QoCMetricValue.id::Int, QoCMetricValue.value::Int,
 QoCMetricValue.creationDate::Date,
 QoCCriterion.id::String,
 QoCMetricValue.modificationDate::Date,
 QoCMetricDefinition.id::String }
```

Listing 1. QoCIM based meta-data specification

```
data ContextInformation = { uri::String, value::String,
 unit::String, creationDate::Date }
```

Listing 2. Context information specification

```
data Message = { context::ContextInformation,
 qoc::[QoCMetaData] }
```

Listing 3. Message specification

Following this formalism, Equation 1 presents the signature of our QoC management functions. The functions have two parameters, one message (m) and different parameters (δ) that are specified in Table II. The functions return one message (m'). The message m' differs from the message m in terms of QoC. The QoC meta-data values of m' are not equal to the QoC meta-data values of m. The QoC management functions do not modify the context information contained in m.

$$f(m{::}\text{Message}, \delta) :\mapsto m'{::}\text{Message} \quad (1)$$

Equation 1: Signature of a QoC management function

Table II presents the name, the behaviour and the parameters of our QoC management functions. The parameters of the functions *addQoCIndicator*, *removeQoCIndicator* and *UpdateQoCIndicator* are specified in Listing 1. The functions *removeQoCMetaData* and *updateQoCMetaData* do not require any parameter.

The function *filterQoCMetaData* is a sophisticated function to remove QoC meta-data following a filter composed of regular expressions. A QoC meta-data value that does not respect the regular expressions specified in the filter is removed. Listing 4 presents the definition of a QoC filter expression. A filter is composed of two elements:

- a list of expressions (`QoCFilterExpression`) to specify constraints on the value of QoC meta-data,
- an operator (`operator`) where the value could be "`UNARY_OPERATOR`" when the filter contains only one expression or a logical operator such as `AND` or `AND` to combine the expressions of the filter.

```
data QoCFilter = QoCFilter {
operator::String, filter::[ QoCFilterExpression ] }

data QoCFilterExpression = QoCFilterExpression {
 qocIdentifier::String, value::String,
 comparator::String }
```

Listing 4. QoC filter definition

A `QoCFilterExpression` has three fields:

- `qocIdentifier`: refers to the variables specified in Listing 1,
- `value`: indicates the expected value of the `qocIdentifier`,
- `comparator`: compares the expected value and the value of the `qocIdentifier`. The accepted values for this field are "==", "! =", ">=", ">", "<" ou "<=".

Listing 5 illustrates a basic QoC filter expression handled by the function. It specifies that the value of the field `QoCMetricValue.value` has to be equal to 60.

```
let filter = QoCFilter{
 operator = "UNARY_OPERATOR",
 filter = [ QoCFilterExpression { value = "60",
  qocIndentificator = "QoCMetricValue.value",
  comparator = "==" } ] }
```

Listing 5. Basic example of QoC meta-data filter

Function	Description	Parameters
addQoCIndicator	Add and compute a QoC indicator in a message	- QoCIndicator.id - QoCCriterion.id - QoCMetricDefinition.id
removeQoCIndicator	Remove a QoC indicator from a message	- QoCIndicator.id - QoCMetricValue.id
removeQoCMetaData	Remove all QoC meta-data	*No parameter*
updateQoCIndicator	Update the value of a QoC indicator	- QoCIndicator.id - QoCMetricValue.id
updateQoCMetaData	Update the value of all QoC meta-data	*No parameter*
filterQoCMetaData	Filter the QoC meta-data of a message	- QoC filter, see Listing 4

TABLE II
INVENTORY OF THE QOC MANAGEMENT FUNCTIONS

B. Context and QoC management functions

When processing context information, QoC meta-data must be processed at the same time. The resulting QoC meta-data actually depend on each specific context management function. This Section presents four functions (aggregation, filtering, inference and fusion) to manipulate context information together with QoC meta-data. They have been selected from the five most cited functions identified in Table I. Storage function does not transform context information and is usually deployed on devices close to the Cloud Computing, as a consequence, this function is not presented in this Section.

Equation 2 formalizes the signature of the functions. Every function has two arguments, a list of messages φ and a set of parameters Δ specific to each function. The functions produce a list φ', possibly empty, of messages. The resulting messages contain new context information and QoC meta-data.

$$F(\varphi{::}[\text{Message}], \Delta) :\longrightarrow \varphi'{::}[\text{Message}] \quad (2)$$

Equation 2: Signature of a context and QoC management function

Independently of any function, Equation 2 highlights an important configuration point: the size of φ. This parameter determines the number of messages that a function have to handle for each its execution. Because the capacities of context sources and the needs of applications constantly change, the number of messages handled by a function could be never

reached or achieved too frequently. In the first case, the function is never executed, in the other case the function may consume too much hardware resources (CPU, RAM...).

To levelling this problem, we integrate in the configuration of a function two parameters:

- the number of messages handled by the function;
- the maximum elapsed time between two executions of a function.

The second parameter offers a guarantee that a function is ever executed after a predetermined time. Our solution supports the expression of whether only one of the parameters or both.

1) The aggregation function: Inspired from the mathematical operator of aggregation [16], we consider the aggregation function with a similar behaviour: it produces a new message from a set of messages. The resulting information has the same abstraction level as the processed information. For example, a function is deployed to aggregate pollution measurements. The resulting information is another pollution measurement.

Some QoC indicators are used to qualify the information processed by the function and the resulting information. So, if the precision and freshness indicators are associated to the pollution measurements aggregated by the function, the same indicators are associated to the resulting pollution measurement.

In this work, we establish a distinction between temporal and spatial aggregation. Temporal aggregation handles information coming from a single context source and produced during some amount of time. Spatial aggregation handles information coming from many context sources that produce the same type of context information periodically.

2) The filtering function: The filtering function has the same objective as the function *filterQoCMetaData* presented in Section IV-A: reducing the amount of information handled by the distributed context manager. With that purpose, the function decides for each processed messages whether they are eliminated or not.

The arguments of the function is then a set of conditions, formulated with regular expressions, relative to the value of context information and QoC meta-data of the message. For example, if the pollution measurement contained in the message is less than 600 parts per million (ppm) and the value of the freshness QoC indicator is more than 45 seconds, drop the message.

3) The inference function: This function produces from a set of data of the same type a new piece of context information of a different type, for example deducing whether it is cold or hot following temperature measurements. The information produced is of a higher level than the information used. It may use different methods that can be based on probability, statistics or inference logical rules. The choice depends mainly on two factors: (1) the computing power of the capsule which performs the function. Some methods require more power than others ; and (2) the type of the information. Indeed, it is common to use statistical operators on digital information and raw measurements, whilst using methods based on inference rules for proposals or statements.

Function (F)	**Characterisation**	
	Configuration parameters	**QoC dependencies** (f)
Spatial and temporal aggregation	*Applies an aggregation operator onto a list of messages. The result is only one message with the same abstraction level.*	
	- Context aggregation operator - QoC management strategy - Optional: QoC aggregation operator	- addQoCIndicator
Filtering	*Analyses the message and decides to remove it or not. The content of the message is never modified.*	
	- Condition about the content of the filtered message - Condition related to the content of the previous filtered messages	- filterQoCMetaData
Inference	*Applies an inference operator onto a list of messages. The result is only one message with a higher abstraction level.*	
	- Inference operator - QoC indicator to add into the QoC meta-data	- addQoCIndicator
Fusion	*Executes a set of functions sequentially or in parallel. The result is a list of messages with a higher abstraction level.*	
	- Ordered list of functions with their configuration	- All QoC functions referenced in Table II

TABLE III
INVENTORY OF CONTEXT MANAGEMENT FUNCTIONS WITH THEIR DEPENDENCIES TO QOC MANAGEMENT FUNCTIONS

The abstraction level of QoC meta-data have to follow the abstraction level of context information. As a consequence, high level QoC meta-data have to be produced from low level QoC indicator associated to the processed context information. To realized that, the QoCIM meta-model offers a solution to specify a hierarchy of QoC indicator, from primitive to composite indicators. For example, from a set of pollution measurements and primitive QoC indicators, as the freshness and precision, the function estimates the corresponding Air Quality Index[1] as context information and the confidence of the index as QoC meta-data.

4) The fusion function: To determine the behaviour of the fusion function, our work is based on the generic JDL software framework proposed by [17]. The authors defined data fusion with five major steps:

i) *source pre-processing* : sorting and grouping received data;
ii) *object refinement* : construction of objects representing the entities observed;
iii) *situation refinement* : detection of situations on detected objects;
iv) *threat refinement* : inference about upcoming events;
v) *process refinement* : monitoring and adjustment process of the previous steps.

The JDL framework is generic and adaptable to various types of applications [18], [19]. In this paper we consider the fusion function as a special function that applies sequentially other functions, namely aggregation, filtering and inference. As a consequence, the transformations applied on context

[1]https://airnow.gov/index.cfm?action=aqibasics.aqi

information and QoC meta-data by the fusion is the result of transformations processed by the other functions.

This section presents a range of functions for processing context information. Some produce new information from a collection of data while others allow to store information for later use. These functions can be configured to determine what computing method to use and also to indicate the number of messages to be actually taken as input. As a result, Table III describes the functions and highlights their dependencies with the QoC management functions. The configurability of the functions opens the way to specify the transformation functions executed by a capsule. Next Section illustrates the declarative solution that we propose to declare the transformation processes handled by a capsule.

V. Self-configurability of capsules

With the purpose to easily setting up and update the configuration of the transformation functions executed by a capsule, we propose a declarative solution. Such a solution becomes possible with a formal definition of the functions and their parameters. As a result we developed a tool based on the functions identified in Section IV. The tool manipulates the configuration of a capsule and stores it into XML document. Based on these document we developed a prototype of configurable capsule. Section V-A details this prototype. Section V-B introduces a new type of capsule that we designed to change at runtime the configuration of transformation functions. Finally this prototype have been improved to become self-reconfigurable. This capsule is presented in Section V-C. It is able to automatically change the configuration of the transformation functions it executes following the current available hardware resources.

A. Configuration

Listing 6 is an example of XML configuration document of a capsule. We developed a dedicated tool that handles available functions with their parameters and QoCIM based QoC criteria to produce this kind of XML document. In the example two functions are specified. The first one is an aggregation that computes the means, the second one is a function to add a QoC indicator into the QoC meta-data of the information produced by the first function. The configuration also indicates the first function is executed as soon as the capsule received 20 messages or every two seconds. The indicator used by the second function is identified with the attributes `QoCIndicator.id` `QoCMetricDefinition.id` `QoCCriterion.id` introduced in Listing 1.

Thanks to the XML configuration documents, we implemented a capsule with an algorithm to automatically set up and deploy the functions specified in the configuration file when the capsule is started. This generic algorithm is composed of the following steps:

1) Configuration file analysis;
2) Create an initialisation function to provide next transformation functions on the messages received by the capsule;
3) For each function declared in the configuration file :
   3.1) Create a buffer to temporally store the messages that will be handled by the function;
   3.2) Create the function with its parameters;
   3.3) Configure the previous function to supply the buffer with its resulting messages;
4) Create a final function to publish the resulting messages to the other capsules;

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Configuration>
 <functions>
  <function id="0">
   <type>ICDFMFunction</type>
   <name>Aggregation</name>
   <operator name="Mean" />
   <nbMessages>20</nbMessages>
   <time>2000</timer>
  </function>
  <function id="1">
   <type>IQoCManagementFunction</type>
   <name>AddQoCIndicator</name>
   <parameters>
    <QoCIndicator.id>14</QoCIndicator.id>
    <QoCMetricDefinition.id>14.1</QoCMetricDefinition.id>
    <QoCCriterion.id>[14.1]</QoCCriterion.id>
   </parameters>
  </function>
 </functions>
</Configuration>
```

Listing 6. Example of an XML Configuration document

Capsules deployed at the edge network have to deal with fast and plentiful context sources and applications. In addition, their needs and capabilities vary quickly and frequently. Moreover, hardware resources available for the capsules are limited, that reduce their efficiency and even making them totally inoperable. To tackle this problem some solution consider offload code as identified [4]. This solution introduces open questions concerning the security and integrity of the capsules, the confidence of the offloaded code and the time required to offload code between entities. We propose in following Sections another approach based on the reconfiguration of the transformation functions executed by the entities. The purpose of our solution does not consist in substitute offloading code but proposing an alternative before offloading code by adapting the configuration of the transformation functions.

Next Sections present our solution to modify at runtime the XML configuration file of a capsule for adapting the behaviour of its functions to the available resources and response time requirements as in Fog Computing. Section V-B details our implementation of a reconfigurable capsule while Section V-C introduces an self-reconfigurable capsule.

B. Reconfiguration

Our Java implementation uses the Apache Common Configuration[2] to detect modifications of the configuration file and automatically restart the algorithm above. Then, measures have been conducted to estimate the elapse time between a modification of the configuration, for example a modification of the value of the attribute `nbMessages`, and the real

[2]http://commons.apache.org/proper/commons-configuration

modification of the function parameters within the capsule. Figure 2 illustrates our measurements made on a desktop with a single core processor. The results are the mean of 20 reconfigurations of the capsule is about adding, removing or modifying functions. The results indicate a capsule can be totally reconfigured under one second and moreover most of the time is used to wait the detection of a new configuration.

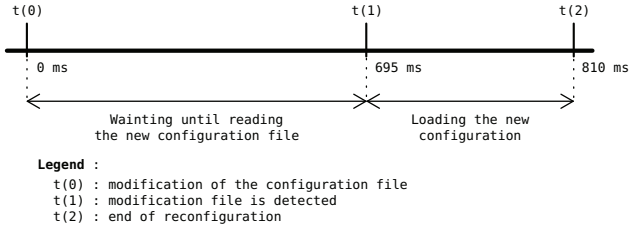

Fig. 2. Average reconfiguration time of a capsule

At this step it is possible to reconfigure at runtime the configuration of a capsule and then change the functions it executes. The results also indicate the reconfiguration process takes less than one second that we assume is acceptable for Fog Computing dynamicity. However, human administrators are not able to take over tens or hundreds entities in the same time. So, this task must be realized by programs that monitor various indicators and decide whether to reconfigure a capsule or not and chose the most appropriate configuration.

With that purpose, we developed a new prototype of a capsule able to self-reconfigure its context processing functions. Next Section illustrates this prototype. This new version of the capsule is now able to decide by itself whether its configuration have to change or not thanks to the monitoring of its hardware resources consumption.

C. Self-Reconfiguration

Our prototype of self-reconfigurable capsule monitors hardware resources usages coming from various logs produced by the entities itself. The logs consist in timestamped data produced regularly by the capsule concerning the tasks performed and hardware resource consumption. To realize the prototype, a Monitor Analyse Plan Execute (MAPE) loop as defined by [20] for the autonomic computing domain, has been integrated into a capsule. The MAPE loop monitors the logs produced by the capsule and decide whether and how to change its configuration or not.

The module “Monitor” collects the logs produced by the capsule about its internal status. Different type of logs are collected concerning, for example, the RAM or CPU utilisation rate of the machine where the capsule is deployed, notification about new incoming messages or the execution of a transformation function.

All these logs are read and interpreted by the module “Analyse”. This module is able to detect hardware consumption peaks under-utilizing of available resources. This kind of events is then forwarded to the Plan module.

The module “Plan” decides in reaction to the event it receives whether the configuration of the capsule has to be changed or not. The module can decide, for example, to change the value of the attributes `nbMessages` or `time`, to activate or deactivate a QoC management function. Another option is to change the operator used by a context management function. When many hardware resources are available, the module can decide to use a more advanced operator on contrary, when very resources are available, the module will decide to use a simple operator. To select the most appropriate operator the Plan module has to be provided with the knowledge that gives for each operator the resource consumption profile.

Finally, the module “Execute” modifies the configuration file of the capsule following the orders of the Plan module. If the configuration file is modified, the reconfigurable capsule presented in Section V-B detects the modification and changes the transformation functions it executes.

We conducted experimentations of the MAPE loop on a desktop (Intel Xeon Processor 5130, 2 Go RAM) and a Raspberry pi 1 model B[3] machine to observe its ability to reconfigure the transformation functions. The study teaches us two lessons. Firstly, both machines successfully support our Java implementation of an self-reconfigurable capsule. Secondly, a difficulty appears concerning the reconfiguration decisions process and more precisely, about the selection of the most appropriate configuration regarding the hardware resource uses.

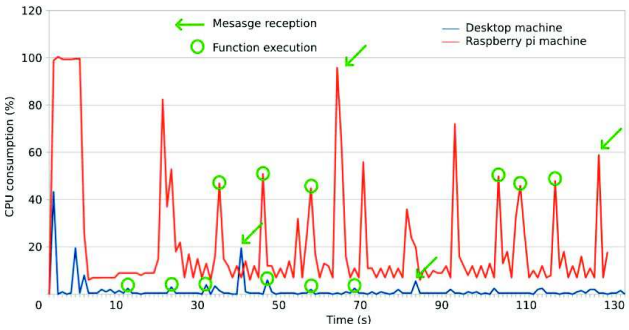

Fig. 3. Average CPU consumption of the aggregation function

We started an evaluation of the resource consumption of the functions with the purpose of supplying knowledge to the Plan module. Our objective is to provide a profile of each function and then be able to anticipate the hardware resource uses following an expected configuration of a capsule. Figure 3 illustrates our first measurements that we realized with the JProfiler tool[4] to get the CPU utilization rate of the aggregation function with a mean operator. As indicated in the Figure, measurements have been made with a desktop machine, with a single core processor and a Raspberry pi machine. In the Figure, two types of peaks are recognizable, some peaks occur when the capsule receives a new message, other peaks occur

[3] www.raspberrypi.org/products/model-b/

[4] www.ej-technologies.com/products/jprofiler/overview.html

when the function is executed. The behaviour is observable with both machines we used. This result constitutes a first element to predict the hardware resource consumption of the configuration of a capsule. We plan to complete the result with an evaluation of the other functions. The result completes a first study that we realized in [21] concerning the overhead of QoC-based filters.

VI. Conclusion

Works concerning context management mainly integrate their own processing functions with their own vocabulary and definition. After a definition of a set of QoC management functions, this article proposes a specification of the most popular context transformation functions with an analysis of their dependencies with QoC processing. Thanks to our specification, we first developed a prototype of a configurable capsule. Our solution provides developers of capsules with a tool to declare the transformations on context information and QoC meta-data their entities have to execute. We improve the flexibility of capsules by enabling their reconfiguration at runtime in order to change their internal settings. Our evaluation indicates that it requires less than one second to change the configuration of our implementation of a capsule. Thanks to this result, we propose a new type of self-reconfigurable capsule able to decide by itself when and what to change concerning its context transformation processes. Our solution constitutes an alternative to offloading code that is currently proposed to tackle dynamicity, volatility and real-time needs of applications based on the Internet of Things.

Because the configuration of a capsule is open and many elements may be changed, a new challenge arises for developing smart "Control Plan" modules able to react as good as possible to the evolution of the hardware resources usage. We therefore plan to perform additional evaluations concerning the hardware consumption of the functions.

The source code of our prototypes is available at: https://fusionforge.int-evry.fr/www/qocim/.

Acknowledgment

This work is part of the French National Research Agency (ANR) project INCOME (ANR-11-INFR-009, 2012-2015). The authors thank Elliot Felgines for his contribution to this work.

References

[1] A. K. Dey, G. D. Abowd, and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Hum.-Comput. Interact.*, vol. 16, no. 2, pp. 97–166, dec 2001. [Online]. Available: http://dx.doi.org/10.1207/S15327051HCI16234_02

[2] EPoSS, *Internet of Things in 2020*, http://www.smart-systems-integration.org/public/internet-of-things, 2008, last access in June 2015.

[3] J.-P. Arcangeli, A. Bouzeghoub, V. Camps, M.-F. Canut, S. Chabridon, D. Conan, T. Desprats, R. Laborde, E. Lavinal, S. Leriche, H. Maurel, A. Péninou, C. Taconet, and P. Zaraté, "INCOME – Multi-scale Context Management for the Internet of Things," in *Ambient Intelligence*, ser. LNCS. Springer, 2012, vol. 7683, pp. 338–347. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-34898-3_25

[4] S. Yi, C. Li, and Q. Li, "A Survey of Fog Computing: Concepts, Applications and Issues," in *Workshop on Mobile Big Data (Mobidata)*. New York, NY, USA: ACM, 2015, pp. 37–42. [Online]. Available: http://doi.acm.org/10.1145/2757384.2757397

[5] Ahmed Banafa, "Iot: A fog cloud computing model," https://www.bbvaopenmind.com/en/iot-a-fog-cloud-computing-model/, 2015, last access : April 2016.

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16. [Online]. Available: http://doi.acm.org/10.1145/2342509.2342513

[7] M. Fanelli, L. Foschini, A. Corradi, and A. Boukerche, "Qoc-based context data caching for disaster area scenarios," in *Communications (ICC), 2011 IEEE International Conference on*, June 2011, pp. 1–5.

[8] P. Nurmi and P. Floréen, "Reasoning in Context-Aware Systems, Mobilife FP6 project, Position paper," http://www.cs.helsinki.fi/u/ptnurmi/papers/positionpaper.pdf, 2004.

[9] S. Sehic, F. Li, S. Nastic, and S. Dustdar, "A programming model for context-aware applications in large-scale pervasive systems," in *IEEE 8th Int. Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct 2012, pp. 142–149.

[10] J. Filho and N. Agoulmine, "A quality-aware approach for resolving context conflicts in context-aware systems," in *IFIP 9th Int. Conf. on Embedded and Ubiquitous Computing (EUC)*, Oct 2011, pp. 229–236.

[11] A. Manzoor, H.-L. Truong, and S. Dustdar, "Quality of context: models and applications for context-aware systems in pervasive environments," *The Knowledge Engineering Review*, vol. 29, pp. 154–170, 3 2014.

[12] P. Bellavista, A. Corradi, M. Fanelli, and L. Foschini, "A survey of context data distribution for mobile ubiquitous systems," *ACM Computing Surveys*, vol. 44, no. 4, pp. 24:1–24:45, Sep 2012. [Online]. Available: http://doi.acm.org/10.1145/2333112.2333119

[13] 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, 2014.

[14] J. B. Filho and H. Martin, "A generalized context-based access control model for pervasive environments," in *2nd ACM SIGSPATIAL Int. Workshop on Security and Privacy in GIS and LBS*. New York, NY, USA: ACM, 2009, pp. 12–21. [Online]. Available: http://doi.acm.org/10.1145/1667502.1667507

[15] P. Marie, T. Desprats, S. Chabridon, and M. Sibilla, "Modeling and using context: 8th international and interdisciplinary conference, context 2013, annecy, france, october 28 -31, 2013, proceedings," P. Brézillon, P. Blackburn, and R. Dapoigny, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 302–315.

[16] M. Detyniecki, "Mathematical aggregation operators and their application to video querying," Ph.D. dissertation, University Pierre et Marie Curie, 2000.

[17] D. Hall and J. Llinas, "An introduction to multisensor data fusion," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 6–23, January 1997.

[18] B. Khaleghi, A. Khamis, F. O. Karray, and S. N. Razavi, "Multisensor data fusion: A review of the state-of-the-art," *Information Fusion*, vol. 14, no. 1, pp. 28 – 44, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1566253511000558

[19] C. Kuka, A. Bolles, A. Funk, S. Eilers, S. Schweigert, S. Gerwinn, and D. Nicklas, "Salsa streams: Dynamic context models for autonomous transport vehicles based on multi-sensor fusion," in *IEEE 14th Int. Conf. on Mobile Data Management (MDM)*, vol. 1, June 2013, pp. 263–266.

[20] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan 2003.

[21] L. Lim, P. Marie, D. Conan, S. Chabridon, T. Desprats, and A. Manzoor, "Enhancing context data distribution for the internet of things using qoc-awareness and attribute-based access control," *Annals of Telecommunications*, vol. 71, no. 3, pp. 121–132, 2016. [Online]. Available: http://dx.doi.org/10.1007/s12243-015-0480-9