A Metadata Monitoring System for Ubiquitous Computing

Caio Batista, Gustavo Alves, Everton Cavalcante, Frederico Lopes, Thais Batista
UFRN – Federal University of Rio Grande do Norte
Natal, Brazil
{caiosergiobatista, gustavoalvescc, evertonranielly, fred.lopes}@gmail.com, thais@ufrnet.br

Flávia C. Delicato, Paulo F. Pires
UFRJ – Federal University of Rio de Janeiro
Rio de Janeiro, Brazil
{fdelicato, paulo.f.pires}@gmail.com

Abstract—In the highly dynamic context of ubiquitous systems, applications need to be continuously aware of QoS and QoC metadata to ensure their required level of quality. We present QoMonitor, a metadata monitoring system that receives synchronous and asynchronous requests from clients (a middleware system that supports ubiquitous applications), recovers metadata from several context providers, and sends them to the clients. We also present an evaluation of QoMonitor under a qualitative perspective, which aims to address the time for assessing QoS and QoC parameters and the time to completely reply to synchronous and asynchronous requests in the context of a health care application. The proposed monitoring system enables ubiquitous applications to focus on addressing the business requirements of the application and abstract away the burden of dealing with the complexities related to synchronous and asynchronous metadata monitoring.

Keywords — metadata; monitoring; Ubiquitous Computing; health care application.

I. INTRODUCTION

Ubiquitous Computing [1] uses a variety of devices, sensors and networks to form a distributed, highly heterogeneous environment integrated to daily activities of users. Typically, ubiquitous applications are composed of services and use context information [2] from several sources to perform their tasks. In this scenario in which applications encompass contextual data and services from different sources, it is essential to know the quality of the provided information and services so that applications can use those that satisfy their requirements. Therefore, the selection of the proper services among those provided by several available providers is performed according to the quality of context information, called Quality of Context (QoC) [3] and/or the quality of the provided services, called Quality of Service (QoS). During the execution of the applications, it is also necessary to ensure that services and context information continue to satisfy the QoS/QoC application requirements.

Both QoS and QoC quality parameters are typically described by metadata, which contain information about observable variables regarding services and/or context, such as resolution, precision, and freshness, for QoC, and error rate, uptime, and response time, for QoS. Ubiquitous applications are inherently dynamic since they use: (i) mobile devices, which can often be or not be in the area covered by a given network; (ii) wireless connections, which are subjected to interruptions and fluctuations in the intensity of the transmitted signal; and: (iii) physical parameters, such as temperature, pressure, location, which can frequently change. In this highly dynamic context, applications need to be continuously aware of QoS and QoC metadata to ensure their required level of quality. For instance, in health care applications, vital data from patients (context information) need to be provided at a high refresh rate (QoC parameter) and by a service with a low response time (QoS parameter).

In this scenario, an important challenge is to provide efficient means to monitor QoS and QoC metadata, thus enabling the application to periodically gather the monitored metadata and also to be asynchronously notified whenever a given metadata becomes available. In the literature, some works on monitoring metadata in ubiquitous applications focus just on QoS [13] or QoC monitoring [12, 14] and either on synchronous or asynchronous mode [12, 14], however, it is important to support both monitoring modes and QoS and QoC metadata. In this perspective, this paper presents QoMonitor, a metadata monitoring system that receives synchronous and asynchronous requests from clients (ubiquitous applications and/or middleware), recovers metadata from context providers, and sends them to the clients. By using the proposed monitoring system, ubiquitous applications can focus on addressing fundamental problems of the application and abstract away the burden of dealing with the complexities related to synchronous and asynchronous metadata monitoring. Furthermore, metadata monitored by QoMonitor can be available to ubiquitous applications, besides it can be associated with a middleware that would be responsible for managing this information in order to select the services that will be used by an application, for example.

The QoMonitor monitoring system consists of three repositories: (i) a metadata repository, which persists all QoS and QoC metadata of the monitored services and the metadata provided by the services providers; (ii) a service repository, which stores information about all monitored services and the parameters needed to communicate with them, and; (iii) a client repository, which stores client information, thus enabling the monitor to communicate with its clients. In addition, QoMonitor contains: (i) an ontology module, which is responsible for specifying metadata using an ontology model to represent the concepts in an unambiguous way; (ii) a requests handler, which receives the requests from clients, gathers the metadata and replies to them, and; (iii) an assessment module, which is responsible for effectively monitoring and assessing QoS/QoC metadata of the services stored in the service repository.
This paper is structured as follows. Section II briefly describes a health care application which serves as a running example used along this paper. Section III presents QoMonitor, the proposed metadata monitoring system. Section IV contains an evaluation of QoMonitor. Section V presents related work. Finally, Section VI contains the final remarks.

II. CASE STUDY

This case study is an application related to the health care context in a scenario inspired in the work of Hegering et al. [4]. The application considers as users patients with critical diseases, doctors, and ambulance staffs that use conventional or specific-purpose mobile devices, connected through heterogeneous wireless networks (e.g. Wi-Fi, 3G, Bluetooth, etc.) and/or wired infrastructures. Patients have their vital functions (e.g. blood pressure, cardiac beat rate, etc.) continuously monitored by body sensors. Besides the information provided by these sensors, medical information (called medical profile) about the patients (e.g. if a patient is smoker or not, if he has diseases and/or allergies, etc.) and previous events/medical diagnosis is available. In health care applications, if a patient has complications in his/her current health state, a set of actions must be taken, such as to trigger emergency staffs to give aid to the patient. This application was chosen because of its relevance in a real-world scenario and because it uses different types of context information deriving from different types of mobile or specific-purpose devices. In addition, many kinds of services can be considered, including services with the same functionality (such as GPS, 3G, or Wi-Fi service location), allowing us to monitor similar services implemented using distinct technologies and providing different levels of quality.

To exemplify a health care application, the blood pressure of a patient was chosen as a parameter to be monitored. If the blood pressure of the current patient is higher than a specific limit (maxBloodPressure), he/she may be suffering a cardiac attack or other kind of complication. Thus, emergency staffs are triggered and information about the current health conditions and the patient’s medical profile are provided, as well as information about his/her localization. At the same time, the application sends a message to the patient’s doctor. Figure 1 gives an overview of the different types of users and the context information processed by the application.

This application consumes services supported by different service providers. MonitoringProvider is responsible for synchronously or asynchronously providing the value of the blood pressure of the monitored patient. AmbulanceManagementSystem manages ambulances, providing services for selecting and triggering emergency staffs. MedicalProfileDatabase provides information about the patient’s medical profile, which contains previous and current information about the patient that may influence in his/her treatment or be correlated to the problem in question. GPSLocalizationMiddleware and CellularLocalizationMiddleware are responsible for providing services about the localization of people and ambulances. Since localization can be provided by several services with different technologies, one of them can be selected to be used by the application according to their quality (QoS/QoC) parameters. Finally, GSMSystem is responsible for sending alert messages (SMS messages) about the current medical state of the patient to his/her doctor.

To sum up, if the value of the blood pressure exceeds the acceptable limit given by maxBloodPressure then the patient’s doctor is alerted about the patient’s medical state. Afterwards, the patient’s localization is obtained by the localization services. Together with the data obtained by consulting the patient’s medical profile, the application chooses and triggers emergency staffs based on the localization of the ambulance staffs with respect to the localization of the patient. Next, the best route (the shortest or the faster course) between the patient’s location and the emergency staff is determined, so that the emergency staff can reach the patient and carry him/her from the current location to the hospital, completing the aid action. Finally, the patient’s medical profile is updated with the event that just took place.

III. QOMONITOR: A METADATA MONITORING SYSTEM

QoMonitor is a metadata monitoring system that receives synchronous and asynchronous requests from clients (ubiquitous applications and/or middleware), recovers metadata from context providers, and sends them to the clients. This Section presents the architecture and operation of QoMonitor (Section III.A) and how it can be used in the context of our running example (Section III.B).

A. Architecture and operation

Figure 2 illustrates the architecture of QoMonitor, which was specified with a modular design, so that each component can work in an independent way. QoMonitor provides two communication interfaces: IClient for communicating with clients, and IServer for communicating with service providers. The Server Façade modularizes all monitor communication with service providers thus being responsible for registering new services in the Service Repository and communicating with the providers. When one of the service providers is registered in QoMonitor, the Server Façade receives the data provided by this provider and forwards them to the Service Repository. The Service Repository is responsible for storing information regarding all monitored services and the parameters required to communicate with them. There are two ways to add new services to the Service Repository. In
the first one the client makes a request to retrieve QoS/QoC metadata, and if the service’s data are not in the repository, then the Client Façade provides the data for storing the new service in the repository. In the second way, the service registers itself in the monitor through the interface provided by the Server Façade. Whenever a new service is added to the repository, the Assessment Module is notified to start the monitoring and assessment of the services.

The Client Façade is responsible for allowing the communication of the clients with the monitor, which can be any ubiquitous application or middleware that needs to make use of QoS/QoC parameters. To perform this communication with clients, it was defined a simple API (summarized in Table 1) that implements the IClient interface. Through this API, clients can register themselves on the monitor and make synchronous and asynchronous requests. To register itself, the client calls the register method, which receives as parameters the client’s name, IP address, and access port. Then, the Client Façade forwards these data to the Client Repository for storing them to be used when the monitor needs to reply the requests of this client. To perform a synchronous request, the client calls the getServiceQuality method, which receives as parameters the data regarding the monitored service and a list of quality parameters to be sent to the client, and returns the parameters with their respective values represented in the ontology format. For instance, considering the GSMSystem service from the case study, a call to the getServiceQuality method would receive the following parameters: <GSMSystem, 192.168.0.100, 8080> (service name, IP address, and access port), <SMS, Person> (a list of input parameters), <errorRate, greaterThan, 0.0> (a list of quality parameters to be monitored), and <errorRate, uptime, responseTime> (a list of the quality parameters to be monitored).

If the client wants to make an asynchronous request, then it should call the subscribeServiceQuality method, which receives as parameters: (i) the data regarding the service and the client; (ii) a list of quality parameters to be sent to the client, and; (iii) a return condition in the form of a <parameter, comparison, value> triple. For instance, if the client wants to be informed when the parameter errorRate of the GSMSystem service is greater than 0%, a call to the subscribeServiceQuality method would receive the following parameters: <GSMSystem, 192.168.0.100, 8080> (service name, IP address, and access port), <errorRate, greaterThan, 0.0> (the return condition). To perform this kind of request, the client must implement a method called callback in order to enable the communication between the monitor and the client, so that this method is responsible for receiving the response from the monitor regarding the asynchronous request. The monitor periodically checks if the return condition has been satisfied, and while it is true, the monitor replies to the client providing the parameters with their respective values represented in the form of the ontology, through the callback method. To stop the sending of responses from the monitor, the client calls the unsubscribeServiceQuality method.

With the data regarding the request, the Client Façade forwards them to the Request Handler using references to clients and service providers in the respective repositories. Finally, when the data regarding the performed assessments are available, the Client Façade receives the QoS/QoC metadata in the ontology form together with a reference to the current client. If the current service is not in the Service Repository, then the Client Façade calls the Service Repository for storing its data and the repository notifies the Assessment Module informing that a new service has been added. This module immediately starts the monitoring and assessment of data.

The Metadata Repository is responsible for persisting all QoS/QoC metadata assessed by the monitor and also QoS/QoC metadata provided by service providers. In turn, the Ontology Module is responsible for representing these data in the form of an ontology as depicted in Figure 3, in which QoS and QoC parameters respectively extend the QoS Parameter and QoC Parameter classes defined in the ontology. An ontology is a data model that represents a set of concepts within a domain and the relationships between them [15], thus providing formal expressiveness and avoiding ambiguity in the semantic interpretations of the same information. For instance, Dobson et al. [5] define the QoS pa-

![Figure 2. QoMonitor architecture.](image-url)

### Table 1. Methods of the QoMonitor Communication API.

<table>
<thead>
<tr>
<th>Method</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>register</td>
<td>clients register themselves in the monitor</td>
</tr>
<tr>
<td>getServiceQuality</td>
<td>clients make synchronous requests to get QoS/QoC metadata</td>
</tr>
<tr>
<td>subscribeServiceQuality</td>
<td>clients make asynchronous requests to get QoS/QoC metadata</td>
</tr>
<tr>
<td>unsubscribeServiceQuality</td>
<td>the monitor stops the sending periodic responses to clients</td>
</tr>
</tbody>
</table>
rameter $ROCOF$ (rate of failure occurrence), which has the same definition of the $error$ $rate$ parameter defined by Guo et al. [6] and that is used in this paper as the error rate in a given time interval. This situation can generate an interpretation problem that can be solved by using ontologies. When a monitor component wants to receive metadata in the ontology format, this component provides a reference to the service in the Service Repository and the Ontology Module forwards it to the Metadata Repository, which performs a search and returns the data of the current service. With these data, the Ontology Module performs operations to represent them in the ontology format used by the monitor.

![Ontology used by QoMonitor for representing metadata.](image)

The main component of the monitor is the Assessment Module. This component is responsible for assessing QoS/QoC metadata of the services stored in the Service Repository and monitoring them and is composed by three types of elements: assessors, Blackboard, and Controller. Each assessor is responsible for assessing one specific quality (QoS/QoC) parameter from information gathered through requests to the monitored services by the Assessment Module. This information is: (i) the time spent to complete the request ($CompletedTime$); (ii) if the service was available or not ($isAvailable$); (iii) the instant in which the request was made ($TimeStamp$), and; (iv) the date and time of creation/sensing of the context information provided by the service (if it is a context service), so that this information is important because it enables inferring the age ($Age$) of the context information provided by the service. The Blackboard component incorporates the idea of a shared data repository (the Blackboard architectural style [16]), which is interesting since the assessors of different QoS/QoC parameters use the same aforementioned information to calculate the value of these parameters. Thus, the use of the Blackboard component avoids a large number of requests to the monitored services since, without this element, each one of the assessors would make isolated requests to the services in order to gather the metadata information, thus negatively impacting their performance. To avoid this problem, the Blackboard centralizes this information so that each assessor is able to receive it and calculate the value of the quality parameter to which the assessor proposes to measure. For instance, assessors regarding QoS parameters such as availability and error rate can make use of historical data stored in the Blackboard about the availability of the service to perform the assessment.

The idea of the Controller component is to control the access to the information stored in the Blackboard and the information gathered from the assessment of the parameters, so that the assessors do not know the source of the data that they use to make the assessment, thus modularizing the architecture. The monitoring of services works independently, by using threads, and starts at the time when the monitor is available, so that monitoring and assessment operations are executed while the monitor receives and replies requests. This continuous monitoring is intended to speed up the response time of requests since when a request is made, QoS/QoC metadata are already stored and can be accessed by the Request Handler to reply to the clients.

Finally, the Request Handler is responsible for retrieving QoS/QoC metadata through the Ontology Module and forwarding them to the clients. When a client makes a synchronous request, the Client Façade forwards it to the Request Handler, which retrieves the current data through the Ontology Module and replies to the Client Façade. When an asynchronous request is forwarded to the Request Handler, it monitors if the QoS/QoC data satisfy the return condition informed by the client; in this case, the Request Handler continuously monitors these data in order to identify whether the return condition is satisfied. When the return condition is met, the Request Handler immediately replies to the Client Façade that calls the callback (listener) method implemented by the client. As the monitoring operation is independent of the other operations performed by the monitor, the response time of a request is considerably small since the service has already been monitored and its parameters have been assessed before making the request. An exceptional situation happens when a client makes a request regarding a service that is not present in the Service Repository, so that this service must be added to the repository and then the monitoring is started.

### B. Monitoring service providers

Before starting the monitoring of services, two time intervals need to be defined in the monitor. The first one is called $TimeToRequest$ and is the time interval in which the Assessment Module makes requests to the service providers. The second time interval is called $TotalTime$ and is the time in which information is considered recent. For instance, if the $TotalTime$ is set to ten minutes, then information gathered more than ten minutes ago will be ignored since this information is considered outdated and can interfere in the assessment calculations of the quality parameters.

Next, the Blackboard receives a list of references to the available services from the Service Repository and makes periodic requests (according to $TimeToRequest$) through the Server Façade to the respective service providers using their data (address and list of parameters), thus returning the time spent to complete the request ($CompletedTime$), whether it has been performed successfully. If the request has not been
successful, then the Server Façade throws an exception that is caught by the Blackboard. For the case study previously described, the context services provided by GPSLocalizationMiddleware and CellularLocalizationMiddleware use QoC metadata, so that the age of this information (Age) is also gathered. In the case study under consideration, no service provider provides the QoS/QoC metadata beforehand, so that after each request the Blackboard stores the request data (CompletedTime, isAvailable, TimeStamp, and Age). If the request has failed, then CompletedTime is equal to -1, isAvailable is false, Age is null, and TimeStamp remains the same. If the service provider itself provides the QoS/QoC metadata, then the Blackboard forwards these metadata to the Ontology Module, which builds the representation of these data in the ontology format and then sends them to the Metadata Repository for storage.

With the data stored in the Blackboard, the Controller is called to access the data history of the requests that are in the Blackboard and forward them to each of the assessors. After all assessors finished their assessment and returned the results to the Controller, it forwards the data to the Metadata Repository for storing them. This execution is repeatedly done with a time interval defined by TimeToRequest and is independent of the requests made by the clients since the idea here is that QoS/QoC metadata are already stored before client requests. Thus, the monitor will quickly reply to client requests and can share data whether two clients make requests to the same service. If for some reason the metadata are not available, e.g. when the first monitoring is performed, then the Request Handler remains on standby until the data are available.

The running example outlined in Section II clearly illustrates the importance of using a QoS/QoC metadata monitor system when deciding which service will be used by an ubiquitous application. For instance, the service providers GPSLocalizationMiddleware and CellularLocalizationMiddleware are responsible for providing services to localize the doctors of the monitored patient and ambulances, each one using different technologies and possibly different QoS/QoC parameters. Without monitoring data, the application will not know which service is best suited to be used, in terms of quality parameters. The client (an ubiquitous application and/or a middleware) can find out the QoS/QoC metadata of the services that are available by making synchronous requests to the monitor. It can also decide which is the best time to use a particular service provider by making asynchronous requests, e.g. in situations when the response time is smaller than fifty milliseconds or the freshness is smaller than two seconds, etc.

IV. Evaluation

This Section presents an evaluation of QoMonitor metadata monitoring system proposed in this paper under a quantitative perspective, which aims to address the time for assessing QoS and QoC parameters (Section IV.B) and the time spent to completely reply to synchronous and asynchronous requests to the monitor (Section IV.C). For the purposes of this evaluation, we have used the health care ubiquitous application outlined in Section II. The services used in the case study were implemented as Web services using the Java programming language and the Apache Axis framework [7] and deployed on an Apache Tomcat application server [8] installed in a computer with an Intel® Core™ i7 2.7 GHz processor, 6 GB of RAM memory and Linux Ubuntu 12.04 operating system, which worked as the server to which requests were performed. In the computational experiments, the QoMonitor monitoring system was executed in a computer with an Intel® Core™ i5 2.3 GHz processor, 4 GB of RAM memory and Mac OS X operating system and has performed requests to the services deployed on the Apache Tomcat server installed in the remote server. Aiming to execute such experiments under similar conditions to those observed in a real-world scenario, QoMonitor and the server in which the services were hosted were placed on different networks, so as not completely disregard the influence of the network in the process. In the quantitative evaluations presented in the Sections IV.B and IV.C, we have performed fifteen independent executions for each service and for each of the six parameters listed in Section IV.A, namely: error rate, response time, MTBF, MTTR, uptime and freshness. In these executions, the values chosen for the TotalTime and TimeToRequest times were twenty minutes and five seconds, respectively.

A. QoS and QoC parameters

In Ubiquitous Computing, context information is gathered from several sources, e.g. it can be provided by users, sensed from sensor devices, derived from multiple origins, etc. Buchholz et al. [3] enumerate some QoC parameters, such as precision, correctness, resolution, freshness, etc. For the evaluation performed in this paper, we have considered the freshness QoC parameter, which expresses the information age, i.e. the time elapsed since the information was generated. Thus, if the information is recent, then it will be more reliable since old information may be outdated. We have chosen just this parameter to assess since other QoC parameters such as precision and resolution are provided by the sensors that measure them [9], so that they can not be properly monitored by our monitoring system. Although precision, resolution and other QoC parameters are not assessed by our QoMonitor, if they are published by the service provider, then the monitor can retrieve them and store these metadata in the ontology format.

Similarly, metadata for QoS parameters are associated with the services used by the ubiquitous applications and are intended to identify the quality of the service. Among the various QoS parameters enumerated in the literature [10, 11], we have considered in this evaluation the following five QoS parameters: (i) response time, which is the time elapsed from the instant in which the client performs a request to the instant in which it processes the response message sent by the server; (ii) MTBF, which is the mean time between system failures during its operation; (iii) MTTR, which is the mean time between a system failure and its return to operation (recovery); (iv) error rate, which measures the error rate for data transmission or service operation in a given time, and; (v) uptime, which refers to the operating time (i.e. availability) of a service.
### B. Assessment of QoS/QoC parameters

Table 2 presents the assessment time (in milliseconds) for each of the quality parameters considered for each service of the case study, from which only the localization services SearchClosestDoctorsCel, SearchClosestDoctorsGPS and SearchClosestAmbulancesGPS, and the monitoring services GetBloodPressure and SubscribeBloodPressure are context services (therefore only these services have the QoC parameter freshness). The assessment time of a given parameter is basically the time spent by the respective assessor to make the calculations of the values regarding this parameter after the necessary data are recorded in the Blackboard component.

In Table 2, the average assessment times are reported in the columns labelled as AVG and the respective standard deviations are reported in the columns labelled as STD. As can be clearly seen in Table 2, all the average assessment times do not exceed the order of 1 millisecond, which is very beneficial in the sense that the monitor does not promote a significantly impact in terms of the assessment of the parameters.

### C. Synchronous and asynchronous requests

Table 3 presents the time spent (in milliseconds) by the monitor to completely reply to synchronous and asynchronous requests made by clients regarding the services of the case study, i.e. the time elapsed between the instant when the request is received by the monitor until the instant in which the monitor sends the response, thus encompassing all operations involved in handling this request. As can be observed in Table 3, the average response times (reported in the columns labelled as AVG) are small, so that the time spent by the monitor for receiving requests and replying to them is most influenced by the network than the monitor itself. Therefore, we can conclude the monitor does not promote a significant impact regarding this issue.

### V. RELATED WORK

To the best of our knowledge, works found in the literature focus just on QoS or QoC monitoring and either on synchronous or asynchronous mode, as we have mentioned.

### TABLE 2. ASSESSMENT TIME OF THE CONSIDERED QoS AND QoC PARAMETERS.

<table>
<thead>
<tr>
<th>Services / Parameters</th>
<th>error rate AVG</th>
<th>error rate STD</th>
<th>response time AVG</th>
<th>response time STD</th>
<th>MTBF AVG</th>
<th>MTBF STD</th>
<th>MTTR AVG</th>
<th>MTTR STD</th>
<th>uptime AVG</th>
<th>uptime STD</th>
<th>freshness AVG</th>
<th>freshness STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetBloodPressure</td>
<td>0.211</td>
<td>0.037</td>
<td>0.069</td>
<td>0.018</td>
<td>0.033</td>
<td>0.008</td>
<td>0.030</td>
<td>0.007</td>
<td>0.033</td>
<td>0.006</td>
<td>0.105</td>
<td>0.022</td>
</tr>
<tr>
<td>SubscribeBloodPressure</td>
<td>0.225</td>
<td>0.025</td>
<td>0.116</td>
<td>0.017</td>
<td>0.043</td>
<td>0.008</td>
<td>0.045</td>
<td>0.003</td>
<td>0.041</td>
<td>0.017</td>
<td>0.126</td>
<td>0.030</td>
</tr>
<tr>
<td>ConsultMedicalProfile</td>
<td>0.201</td>
<td>0.028</td>
<td>0.074</td>
<td>0.012</td>
<td>0.036</td>
<td>0.007</td>
<td>0.034</td>
<td>0.004</td>
<td>0.036</td>
<td>0.002</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SearchClosestDoctorsCel</td>
<td>0.251</td>
<td>0.010</td>
<td>0.120</td>
<td>0.006</td>
<td>0.045</td>
<td>0.005</td>
<td>0.042</td>
<td>0.002</td>
<td>0.045</td>
<td>0.029</td>
<td>0.137</td>
<td>0.030</td>
</tr>
<tr>
<td>SearchClosestDoctorsGPS</td>
<td>0.181</td>
<td>0.005</td>
<td>0.067</td>
<td>0.006</td>
<td>0.028</td>
<td>0.003</td>
<td>0.028</td>
<td>0.005</td>
<td>0.031</td>
<td>0.006</td>
<td>0.094</td>
<td>0.007</td>
</tr>
<tr>
<td>SendSMS</td>
<td>0.257</td>
<td>0.024</td>
<td>0.118</td>
<td>0.013</td>
<td>0.044</td>
<td>0.005</td>
<td>0.041</td>
<td>0.011</td>
<td>0.037</td>
<td>0.003</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SearchClosestAmbulanceGPS</td>
<td>0.234</td>
<td>0.023</td>
<td>0.117</td>
<td>0.016</td>
<td>0.041</td>
<td>0.005</td>
<td>0.040</td>
<td>0.004</td>
<td>0.037</td>
<td>0.004</td>
<td>0.133</td>
<td>0.017</td>
</tr>
<tr>
<td>CallAmbulance</td>
<td>0.225</td>
<td>0.034</td>
<td>0.088</td>
<td>0.021</td>
<td>0.034</td>
<td>0.008</td>
<td>0.034</td>
<td>0.008</td>
<td>0.032</td>
<td>0.006</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DetermineRoute</td>
<td>0.233</td>
<td>0.025</td>
<td>0.110</td>
<td>0.014</td>
<td>0.041</td>
<td>0.003</td>
<td>0.039</td>
<td>0.003</td>
<td>0.037</td>
<td>0.002</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>UpdateMedicalProfile</td>
<td>0.184</td>
<td>0.017</td>
<td>0.053</td>
<td>0.012</td>
<td>0.025</td>
<td>0.005</td>
<td>0.025</td>
<td>0.006</td>
<td>0.028</td>
<td>0.004</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### TABLE 3. TIME TO REPLY SYNCHRONOUS AND ASYNCHRONOUS REQUESTS

<table>
<thead>
<tr>
<th>Services / Requests</th>
<th>synchronous AVG</th>
<th>synchronous STD</th>
<th>asynchronous AVG</th>
<th>asynchronous STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetBloodPressure</td>
<td>60</td>
<td>12</td>
<td>65</td>
<td>28</td>
</tr>
<tr>
<td>SubscribeBloodPressure</td>
<td>60</td>
<td>10</td>
<td>64</td>
<td>23</td>
</tr>
<tr>
<td>ConsultMedicalProfile</td>
<td>70</td>
<td>19</td>
<td>57</td>
<td>18</td>
</tr>
<tr>
<td>SearchClosestDoctorsCel</td>
<td>65</td>
<td>16</td>
<td>57</td>
<td>8</td>
</tr>
<tr>
<td>SearchClosestDoctorsGPS</td>
<td>61</td>
<td>11</td>
<td>57</td>
<td>10</td>
</tr>
<tr>
<td>SendSMS</td>
<td>57</td>
<td>8</td>
<td>61</td>
<td>23</td>
</tr>
<tr>
<td>SearchClosestAmbulanceGPS</td>
<td>70</td>
<td>18</td>
<td>60</td>
<td>15</td>
</tr>
<tr>
<td>CallAmbulance</td>
<td>69</td>
<td>19</td>
<td>68</td>
<td>13</td>
</tr>
<tr>
<td>DetermineRoute</td>
<td>65</td>
<td>16</td>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>UpdateMedicalProfile</td>
<td>64</td>
<td>15</td>
<td>53</td>
<td>17</td>
</tr>
</tbody>
</table>
they have a storage module. Since our work focuses specifically on monitoring QoS and QoC metadata and making them available for ubiquitous applications or middleware, the QoMonitor system provides the subsides needed to serve as input to another component (or even a middleware) that would be responsible for triggering these automatic actions associated with changes regarding QoS and QoC parameters.

Finally, Zheng and Wang [14] propose a tool that supports QoC management. Requests are handled by a context reasoner, which filters context information and notifies the subscribed components about context changes, thus supporting asynchronous requests. In addition, services are stored in a context repository and metadata are represented by an ontology. Although this work is very close to our proposal, it does not support synchronous requests and focuses just on QoC monitoring.

VI. FINAL REMARKS

In this paper we presented QoMonitor, a metadata monitoring system that is in charge of handling synchronous and asynchronous requests for monitoring QoS and QoC metadata. QoMonitor recovers metadata from several context providers, uses an ontology to represent such metadata, and sends them to the clients. By using the proposed monitoring system, ubiquitous applications can abstract away the burden of dealing with the complexities related to synchronous and asynchronous metadata monitoring. In addition, these monitored metadata can be available to ubiquitous applications and/or a middleware that would be responsible for managing this information in order to select the services that will be used by an application, for example. We have implemented this system and used it in a health care application and the evaluation of QoMonitor showed that the average time for assessing QoS and QoC parameters and the time spent to completely reply to synchronous and asynchronous requests to the monitor are significantly small. As a future work we aim to evaluate the delay when a given service is not available at the service repository yet and the monitoring system needs to ask a third-party element (typically an underlying middleware) to discover which context provisioning system provides such service.

REFERENCES