CodaQ: A Context-Aware and Adaptive QoS-Aware Middleware for Activity Monitoring

Shahram Nourizadeh, Ye-Qiong Song, and Jean-Pear Thomesse

INPL - Nancy University, Loria, France
{Shahram.Nourizadeh,Song,Thomesse}@Loria.fr

Abstract. For providing real-time data collecting in a telehealthcare system composed of wireless sensor network and home automation network, a middleware called CodaQ is designed. It provides context data and takes into account QoS requirements of the applications. In this paper, we show how the data are modeled for including context information and how the QoS requirements are handled within a middleware. First measurements on a test bed have been carried out showing the good performance of our design.

Keywords: Context-aware Middleware, adaptive QoS, Telehealthcare, Wireless Sensor Networks.

1 Introduction

Activity monitoring (or actimetry) of the older people at home is one of the important applications of the telehealthcare systems. Actimetry needs to collect in real-time sensor data through an efficient communication system with QoS guarantee. Wireless sensor network (WSN) is an emerging technology for building telehealthcare systems and has incited important research efforts [1]. One of the main research topics is to provide the QoS for the sensor data transfer in terms of timeliness and reliability under stringent energy constraints. However how to map the application requirement and the network QoS services needs still further investigation.

In a typical activity monitoring system data traffic flows can be classified into 3 categories with different priorities: 1. Medical sensors; 2. Environmental sensors; 3. Multimedia. Medical data flow has the highest priority and multimedia the lowest one. Data collection can be achieved through a home gateway. Different data collection strategies can be applied in each category: Event-based, periodic or mixed data collection, which demand managing the QoS requirements (by applying priorities) and temporal coherence (time stamp). The QoS requirements of the sensor data are determined by sensor data type and the condition of the patient. For example, the medical information like ECG may require a higher priority than the other sensors when a patient has an unusual heart rate, in this case the home gateway must be able to give the higher priority to ECG data flow. The temporal coherence is needed for multi-sensor data fusion (e.g. related data are collected during a coherent time window) and this can be achieved by time stamping at the home gateway.

Moreover sensor data should be interpreted within their current context. This can be achieved by using context-aware computing. Context is any information about the user and the environment that can be used to enhance the user's experiences [2]. Context-aware systems represent extremely complex and heterogeneous distributed systems, composed of sensors, actuators, application components, and a variety of context processing components that manage the flow of context information between the sensors/actuators and applications [3]. The objective of our work is to deduce the state of the patient at any moment and the contextual information helps us to do this.

2 Related Works

In the literature, several context-aware middleware have been proposed. ConStruct [4] is a knowledge-based pervasive computing middleware that provides semantically unification over a range of home- and web-based automation systems. It is a sensor-fusion-based middleware for smart homes. In ConStruct all data are modeled using the Resource Description Framework (RDF) [5], which provides a standardized way to model contextual information and properties.

Another adaptive middleware design for context-aware applications is presented in [6]. The middleware abstracts the applications from the sensors that provide context. It also aims to implement autonomic properties, such as self-configuration and failure tolerant. The goal in the design of this middleware has been to provide adaptation with good performance. The approach assumes that Context Providers (CPs) are able to estimate their QoC (Quality of Context: precision of the information provided by middleware about the context).

[7] presents the design and implementation of a middleware approach for context-awareness, and adopted fuzzy logic as an intelligent reasoning method for selecting data dissemination protocols in the design of the decision mechanism. The approach uses a context space theory model shown in [8] for modeling the fundamental nature of context and enable context and situation awareness for context processing. Its context model gives a common representation for context that all entities in the environment use of pervasive computing. Instead, it provides a common base on which various reasoning mechanisms can be specified to handle context.

[9] presents MidCASE a middleware to support the context-aware application development for wireless sensor networks. The middleware has five layers and two cross layers: hardware abstract, service registry, context model, reasoning and application presentation layers; energy management module and security module.

Numerous other attempts to build middleware or infrastructure for context-aware systems have been made also, but they have provided only partial solutions; for instance, most have not adequately addressed issues such as QoS, mobility, fault tolerance or privacy. To fully use the benefits of context-aware system, an infrastructure which enables QoS self-adaptation and protocol interoperability among heterogeneous components is required. In these systems, any change in context provision must be detected quickly and adaptation occurs swiftly. This is important in telehealthcare systems that monitor a person with health problems. A number of existing approaches appear to be quite suitable and efficient but none of them appears to be a perfect match for the requirements.

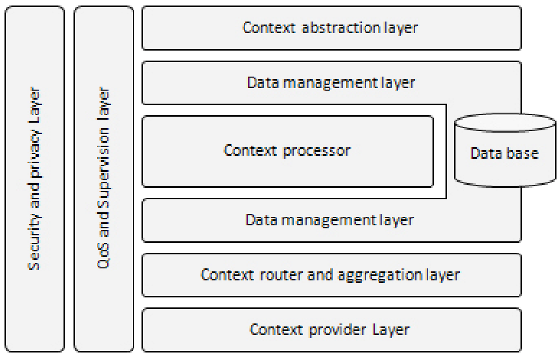
Security and privacy Layer
QoS and Supervision layer
Context abstraction layer
Data management layer
Context processor
Data base
Data management layer
Context router and aggregation layer
Context provider Layer

Fig. 1. Reference model

Table 1. Comparison of surveyed Middlewares

	Service or Layer							
Middleware	**Context abstraction**	**Supervision and QoS**	**Database**	**Context processor**	**Data management**	**Context router and aggregation**	**Context provider**	**Security / privacy**
ConStruct	yes	no	no	yes	no	yes	yes	no
[6]	yes	yes	no	yes	no	yes	yes	no
[7]	no	no	no	yes	no	yes	yes	no
MidCase	yes	no	no	yes	no	yes	yes	yes

In order to consider the challenges we discussed before, we defined a reference model (Fig. 1). This model presents all modules that a Middleware must have, to be able to deal with the challenges and ensure a real context deduction. In this model:

- Context abstraction layer provides an abstraction of the context for context-aware applications.
- QoS and Supervision layer guarantees the supervision of the system and applies runtime QoS need.
- Database saves the events and states detected from the context.
- Context processor layer uses the raw data provided by the context router, to reason the context.
- Data management layer controls the databases and ensures the management of all data that are exchanged between the other layers.
- Context router and aggregation layer is a gateway between the context provider layer (sensors) and the middleware.
- Context provider layer is a distributed application installed on the sensors to detect and send the events to the middleware.
- Security layer is the responsible for providing the security of the system.

Table 1 summarizes the capabilities of the surveyed solutions based on considered reference model and shows that comprehensive solutions do not yet exist. In this paper we present CodaQ, a context- and QoS-aware health and activity monitoring middleware, which is the key building block of a global telehealthcare system. CodaQ introduces fuzzy decision support and addresses a large subset of the requirements listed in Table 1 except security and privacy which are our future work.

3 Data Modeling for Taking into Account the Context

In CodaQ all data are given a uniform representation, and their level of abstraction is raised through the use of data format definitions for *raw event*, *deduced state, virtual sensor, zone, query and query reply* that will be detailed in the following.

The **Raw Event** is the output of a sensor on which no process was accomplished; it is the input of the application. The Raw Event shows an event detected by a sensor on a specific place, like movement or presence of a person, window or door opened or closed, light switched on or off. The Raw Event is presented by **RawEvent** parameter:

RawEvent (Id, Time, SensorID, {Values}, Zone).

For example **RawEvent**(10,22/11/2010-14:15:23,316,21,Room01), means that the raw event number 10 is detected by sensor number 316, at 14h15m23s of 22/11/2010, at Room01 and the measured value is 21.

The **Virtual Sensor** is the main component of hardware abstraction in our middleware. Virtual sensors abstract from implementation details of access to the sensor network and correspond either to data streams received directly from sensors or derived from other virtual sensors accessible through the network (thus a set of virtual sensors). The virtual sensor descriptor in our architecture is presented as:

VirtualSensor(Id, {Sensors}).

Our middleware creates these virtual sensors dynamically based on context state (state-based virtual sensor formation).

The input of a **VirtualSensor** is **RawEvent** and its output is a deduced state which presents the state of the context. The deduced state is the result of the processes in context processor on the raw events, and presents the context's actual situation. The deduced state is represented by **DeducedState** descriptor:

DeducedState (Id, Value, Category, Time, Zone/Place, VirtualSensorId).

For a user, the value of **DedusedState** could be: Fall, Leave/Arrive, Sleep, Eating, Reading etc. For example **DeducedState** (128,"Eating", "Normal", 22/11/2010-16:10:12, "Kitchen", 257), means the people was eating at 22/11/2010-16:10:12 in the kitchen, this state is a normal state and in deduced by virtual sensor which has Id 257.

Zone, presents the location of a given event or state:

Zone (Id, {Sensors}).

Query is the command generated by the system to an actuator or a sensor. Light switch off, change the lighting level, open the door are some examples of a query. It could also be used for the supervision of the system in order to detect any failure in the hardware or the communication links. The query is represented by Query descriptor:

Query (Id, Category, {Sensors}, {Actuators}, Zone, Supervision (Bool)).

If supervision is true, then the answer will be a **QueryReply** which is presented by **QueryReply** descriptor as the acknowledgement of the sensors or actuators to a Query:

QueryReply (Id, QueryId, {Sensor}/{Actuator}),

and if supervision is false, the answer will be a **RawEvent**.

4 Architecture of CodaQ

In our architecture (Fig. 2) it is **Context Collector** (equivalent of Context router and aggregator in reference model) that collects the context raw events from the distributed context providers and saves them in the Raw Event data base. This layer has a Queue Manager sub-layer which controls the input buffer by information received from QoS Specification Provided.

The next layer is the data management layer, which has 2 sub layers:

- **Data Manager:** receives queries from different components of the middleware, executes them and resend the results.
- **Database:** consists of six databases.

Context process layer deduces real context state from raw events (basic sensed context data) using fuzzy rule-based reasoning techniques, and also checks for knowledge consistency in the fuzzy context knowledge base. In addition, temporal and spatial

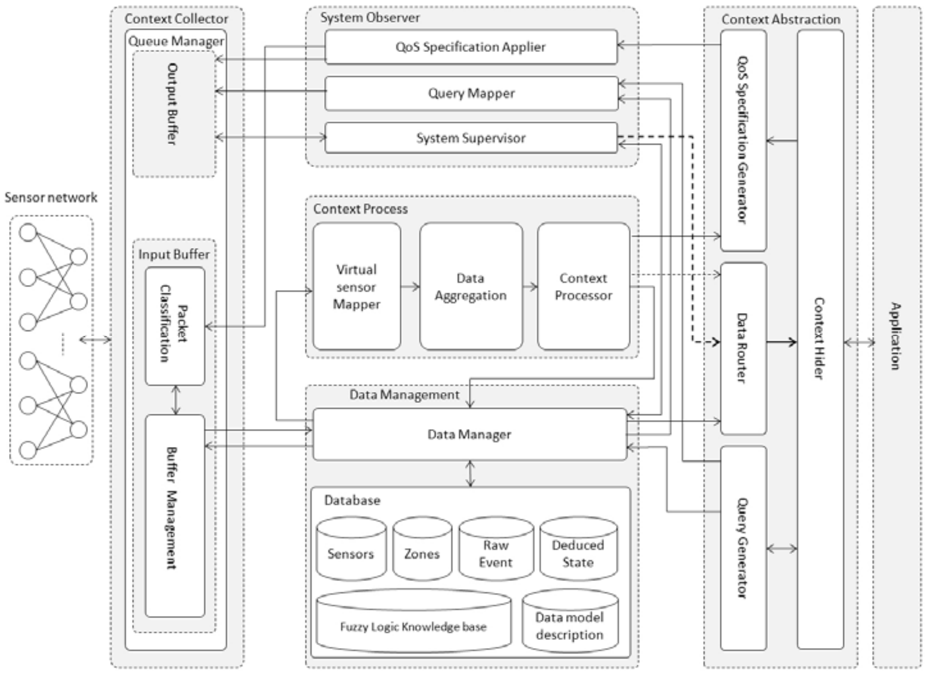

Fig. 2. CodaQ's architecture

validation must be taken into account. Temporal validity is defined as the period of time to which a single instance of context information is applicable, and spatial resolution is defined as the precision with which the physical area, to which an instance of context information is applicable, is expressed [10]. To consider these needs, this layer has three sub layers:

- **Data Aggregator:** aggregates the context information by using Raw Event Database and Virtual Sensor information and provides them to Context Processor Layer. This layer verifies the temporal validity in a given time window. If one or more data are not in the time window, the layer request from Query generator layer to send a query to the sensors in order to provide fresh data.
- **Virtual Sensor Mapper:** provide necessary information about all virtual sensors of the network. It uses Sensor and Zone Databases to create virtual sensors and provides them to the Data Aggregation layer. The concept of Virtual sensor is used to apply the spatial validity.
- **Context Processor:** uses information provided by Data Aggregation layer, processes them in order to deduce real state of the context and saves deduced states in the Deduced State Database.

QoS observer layer is the next layer of the middleware which has three sub layers:

- **QoS Specification Applier**: In our design QoS observer applies QoS in two levels: Embedded and Run-time State-based QoS. For the embedded QoS guaranty, we use the adaptive routing algorithm which optimizes the resource utilization by considering Energy, Mobility, Failure history and Load of the nodes [11]. The Run-time State-based QoS is based on priorities, defined by application in a given time in order to adapt the resource needs of the application. QoS Specification Applier uses run-time information about the context to choose the most suitable configuration of applications and middleware services. This requires that the middleware has a specification of elements in the context, and QoS specifications. Some examples of context information that can be used to improve the QoS are: user's location and user's health state. The Context Collector can then prioritize and schedule the information delivery based on which information elements are most important to the user, and can select information processing based on user characteristics and resources, and allocate the buffer to the most important operations.
- **System Supervisor:** it creates Queries and sends them to the specified destinations (Sensors and Actuators) in order to monitor health state of the installed sensors and actuators and detect any failure in them or the communication links.
- **Query Mapper:** uses Sensors and Zones database and generates the necessary queries in order to provide the latest (fresh) information of the sensors.

Finally, the **Context abstraction** layer uniforms the information and data transferring between Application and the Low level layers (Middleware and Context providers) by applying data model definitions. It has three sub layers:

- **Context Hider:** receives the requests from application and sends it to QoS Specification Generator and Query Generator layers. It also receives results of the queries and sends them to application layer.
- **Query generator:** generates persistent controls on demand of application layer. This layer is the bridge of communication of application layer with the sensors

and actuators and allows applications to extract desired context information from the networks.

– **QoS Specification Generator:** generates QoS Specifier by using QoS need of the application, received from Context Router layer.

5 Implementation and Performance Evaluation

A prototype has been implemented to verify the feasibility of our proposals, based on our laboratory apartment in Colmar. The programming environment of the middleware is Visual Studio .Net 2005 C# and ASP .Net and the service discovery is based on SOAP. The prototype utilizes a SQL server database. The prototype utilizes Imote2 sensors in the sensor network side. The programming environment for these sensors is Imote2.Biulder SDK under Visual Studio .Net 2005 C# and Microsoft .Net Micro Framework 2.5, based on TinyOS 2.0.

We examined the response time of a context query, which is defined as the interval between the time when the query is issued from an application by receiving the data from the sensor and the time when the application receives the query result as a feedback from the actuator. In this scenario, we use Imote2 motes to sense outdoor luminosity. The sensed data will be sent to the gateway and will be processed by the middleware, and based on the result, the indoor luminosity will be varied.

Table 2 shows the time breakdown of a query in our system. The results presented in this table are the average of 10 measurements. Data collection is the time spent on getting the context attribute values from the network of Imotes, Context processing is the time the the context processing takes to evaluate the data and generate the query, Query generation is the time the system spends on creating the query and finally Reply is the time required to send the query result to the application.

In the table we see that, the total response time of a query in our system is less than one second. The collection of context data from the sensor network takes 34% of the total time and the most time-consuming task among the others is the reply time which takes 48% of total time. The time for context processing takes 14% and the query generation time is just 4% of the total response time. If we use direct command to an actuator we do not have the context collection and processing times. That means by automation the home environment by our system, we add an overhead of 273 ms for context collection and 14% for context processing or 48% of the total response time which seems to be acceptable as a cost of automation of the context processing and control.

Table 2. Response time

Task	Time (*ms*)	% in total time
Context collection	378	34%
Context processing	112	14%
Query generation	34	4%
Reply	237	48%
Total time	**797**	

6 Conclusion and Future Work

The experience gained through our work showed us the lack of a bridge between the healthcare applications and the sensor networks in a telehomecare system, to guarantee QoS needs, for supervision of the system and for easy configuration and installation of the sensor and actuators.

For these reasons, we developed CodaQ, a context-aware and priority-based adaptive QoS middleware. In this middleware all data are given a uniform representation, and their level of abstraction is raised through the use of basic data modeling. A context-based adaptive QoS method was implemented on the system, in two levels: Embedded and State-based QoS.

As our future work, we plan to investigate the extended use of our data modeling system to consider a wider class of context data. Moreover, since in many situations available context data may be insufficient to unambiguously determine the activity performed by a user, we are investigating the use of fuzzy ontologies to deal with uncertainty and fuzziness.

References

1. JeongGil, K., Chenyang, L., Srivastava, M., Stankovic, J., Terzis, A., Welsh, M.: Wireless Sensor Networks for Healthcare. Proceedings of the IEEE 98, 1947–1960 (2010)
2. Dey, A., Abowd, G., Wood, A.: CyberDesk: A Framework for Providing Self-Integrating Context-Aware Services. Knowledge-Based Systems, 3–13 (1999)
3. M. C. Hübscher, J. A. McCann, “An adaptive middleware framework for context-aware applications”, Personal and Ubiquitous Computing, Vol. 10, No.1, pp. 12–20, (2006).
4. Henricksen, K., Indulska, J., McFadden, T., Balasubramaniam, S.: Middleware for Distributed Context-Aware Systems. In: OTM Confederated International Conferences, pp. 846–863. Springer, Heidelberg (2005)
5. W3C Resource Description Framework (RDF), `http://www.w3.org/RDF/`
6. Huebscher, M., McCann, A.: An adaptive middleware framework for context-aware applications. Personal and Ubiquitous Computing 10(1), 12–20 (2006)
7. Ye, N., Wang, R., Ma, S., Wang, Z.: Fuzzy Logic Based Middleware Approach for Context Processing. JDCTA 3(3), 36–41 (2009)
8. Padovitz, A., Loke, S.W., Zaslavsky, A., Burg, B., Bartolini, C.: An approach to data fusion for context awareness. In: Dey, A.K., Kokinov, B., Leake, D.B., Turner, R. (eds.) CONTEXT 2005. LNCS (LNAI), vol. 3554, pp. 353–367. Springer, Heidelberg (2005)
9. Bai, Y., Ji, H., Han, Q., Huang, J., Qian, D.: MidCASE: A Service Oriented Middleware Enabling Context Awareness for Smart Environment. In: MUE 2007, pp. 946–951 (2007)
10. Guesgen, H., Marsland, S.: Spatio-Temporal Reasoning and Context Awareness, pp. 609–634. Springer, Berlin (2010)
11. Nourizadeh, S., Song, Y.Q., Thomesse, J.P.: An Adaptive Hierarchical Routing Protocol for Wireless Ad-hoc Sensor Networks. In: IEEE NGMAST 2009, Cardiff, Wales, UK(2009)