A SQL-based Context Query Language for Context-aware Systems

Penghe Chen, Shubhabrata Sen, Hung Keng Pung and Wai Choong Wong
National University of Singapore
Singapore
e-mails:{g0901858, g0701139, dcsphk, elewwcl}@nus.edu.sg

Abstract—**Context-aware computing is a typical paradigm of ubiquitous computing and aims to provide context information anywhere and anytime. Context data management which handles gathering, processing, managing, evaluating and disseminating context information is the heart of context-aware system. Context provision and acquisition, thus, become crucial for context-aware computing. In order to decouple application developers from the tedious work of managing underlying context data sources, a proper context query language should be defined to express context information requirements without considering details of underlying structure. Different types of context queries have been proposed previously and an evaluation demonstrates that SQL-based and RDF-based query languages are most powerful in expressing context queries. However, although the RDF-based languages are more suitable for expressing relations and reasoning operations, it is not as flexible as the SQL-based methods in representing user requirements. Additionally, the RDF-based language creates a large amount of overheads due to its various definitions of classes, sub-classes and relations. In order to address these issues, we propose and design this SQL-based context query language which is easy to use and very flexible to express queries with different constraints. It supports both pull and push based context data retrieval, as well as different context processing functions to generate meaningful context information.**

Keywords-context; context-awareness; context query language; SQL-based; context data management; context provision and acquisition; ubiquitous computing.

I. INTRODUCTION

The advances of wireless communication and mobile computing technologies have brought people into the era of ubiquitous computing. Context-aware computing which is a paradigm of ubiquitous computing, aims to provide information to people anywhere and anytime. As a result, applications can recognize and adapt to changes in the environment automatically. Context is usually defined as any information that can be used to characterize the users' situations [1], while context-aware systems are systems that can leverage on context information to adapt their behaviors in order to realize context-awareness [15].

An important part of a context-aware system is a context data management system which takes charge of gathering, processing, managing, evaluating and disseminating context information [15]. A properly designed context data management system could decouple the lower level data collection part from the upper layer application design, so that developers are not concerned with the actual details of the underlying data collection. Additionally, context data management systems can perform intermediate context processing to improve the quality of context data. In order to better utilize -systems, application developers should have access to programming tools that abstract the underlying details of the context data collection process.

An important part of this problem is to define and design an appropriate Context Query Language (CQL) to formally represent context data acquisition requirements through queries. A context query language is a formal language for representing queries in context-aware systems and defines the basic structure and syntax for a context query [15]. A properly defined CQL can ease the process of expressing the required context data retrieval conditions as well as the corresponding context query processing operations.

Query languages designed for traditional database systems are not suitable for context data querying due to the special characteristics of context data. First, context data is usually dynamic, which may produce frequent updating operations and easily cause data inconsistency using traditional database system. Additionally, context data is usually not well structured and can be various kinds of information, such as situation information or metadata information, which cannot be properly represented by query languages designed for well structured schema based data. Furthermore, context data can come from heterogeneous and distributed context sources [7], which makes traditional query languages hard to represent the data information. In addition, context data management requires lots of context reasoning operations to derive higher level information [9], which cannot be represented by traditional query languages. A properly designed CQL should take care of these characteristics.

In order to address these issues, different methods have been proposed to design CQL as discussed in [3][6][9]: Structure Query Language (SQL)-based, Resource Description Framework (RDF)-based, Graph-based, Extensible Markup Language (XML)-based and Application Programming Interface (API)-based query languages. By analyzing the advantages and drawbacks of each type of these query languages, the evaluation conducted by Haghighi et al. [6] demonstrates that RDF- and SQL-based CQLs are more powerful and effective. However, compared with SQL-based method, RDF-based method is more specialized for RDF or ontology based context representation, which limits

 ISBN: 978-1-61208-364-3

its capability and makes it hard to be integrated with traditional database data. Additionally, RDF is relatively complex in defining and creating various kinds of classes and subclasses as well as relations, which produce large amounts of overhead and make it difficult to implement. On the other hand, an SQL-like language is very flexible and can significantly reduce the learning curve, providing a developer has worked with standard SQL [19].

In this paper, we discuss the design of a new SQL-based CQL that helps application developers to construct context queries to acquire context data from various domains and different context sources. Our proposed CQL can also support different types of context acquisition manners as well as context operations.

The rest of the paper is organized as follows. Analysis of requirements for a well-defined CQL is given in Section II. Section III presents some discussions on existing CQLs. The proposed SQL-based CQL is described in Section IV. Section V discusses the evaluation and the paper is concluded in Section VI.

II. REQUIREMENTS

SQL is a widely utilized query language in relational database management systems. However, directly utilizing SQL in context data management is not possible as context data has its own characteristics that are different from relational database data. This is why a separate CQL is required to support queries on context data, but the powerfulness and experiences of SQL can be learnt to create a CQL upon the existing SQL structure. Compared to traditional database data, context data has its own special characteristics [6].

- **Dynamic or static**: unlike traditional database data which are usually static, most of context data can be either static or dynamic which makes the management and accessing differently.
- **Stream data**: context data may be queried and accessed in a continuous or periodic manner which is different from traditional database queries usually with specific or discrete data.
- **Temporal and spatial relationship**: different from traditional database data, context data is closely related to spatial and temporal information.
- **Situational information**: compared to simple and clear traditional database data, context data can be derived situational information based on other information.
- **Unstructured**: most of time, context data do not have a predefined uniform schema like that in traditional database data.

All these dedicated characteristics of context data should be carefully considered while designing a CQL. In addition, the inherent nature of context data management can also attribute some requirements on the design of CQL. Some characteristics of pervasive computing environment are summarized by Perich et al. [13]: autonomy, mobility, distribution and heterogeneity, which also create challenges to the design of CQL, just as discussed in [6]. Autonomy implies that each entity is an independent context source. The mobility aspect indicates that entities can appear and disappear frequently from the network. The distribution aspect implies that context data should be retrieved from different entities whereas the heterogeneity aspect results in no common data model. Also, pervasive applications need to access context information about users and their devices without dealing with the details of the data collection process [4]. Additionally, context queries should be expressed in a context model that can be converted into different data models as required.

Based on these considerations, we can list out some of the requirements in designing CQL for context-aware systems as following:

- The CQL should express context queries in an abstract level without indicating details of context sources such as locations and storage mechanism.
- The CQL should express context queries utilizing a predefined context data model and this model should be well integrated with the underlying context data representation
- The CQL should express queries acquiring context data either in a pull-based or push-based manner by specifying filters and conditions.
- The CQL should express certain advance context processing operations to filter and process context data, such as aggregating and reasoning operations.
- The CQL should express context queries being able to access context data from more than one context source as well as continuous data.
- The CQL should be able to express compound queries concerning multiple contexts domains and conditions.

III. EXISTING CONTEXT QUERY LANGUAGES

Different types of CQLs leveraging on different technologies have been surveyed and classified previously: SQL-based, RDF-based, XML-based, API-based and Graph-based CQLs [3][6][9]. The evaluation conducted in [6] has further demonstrated that SQL-based and RDF-based CQLs are more powerful and effective than others. In this section, we will review existing SQL-based CQLs and compare with RDF-based CQLs to illustrate the necessity for a new SQL-based CQL.

A. SQL-based CQL

SQL is a well-established, declarative query language that has been recognized as an effective language for accessing traditional database. As demonstrated in [6], SQL-based CQL is one of the two most effective types of query languages in context-aware computing. Various SQL-based CQLs have been proposed and designed previously and we are inspired by these works to design our new SQL-based CQL.

Contory [18] utilizes a SQL-based CQL as an interface to provide context information for applications. Contory supports both pull- and push- based methods to acquire context data, but it does not support context processing operations and has poor performance on consolidating context information from different sources. Another SQL-based query language designed to query data in an ambient

 ISBN: 978-1-61208-364-3

intelligent environment is presented by Feng [2], which uses context information to define data retrieval conditions in a relational database. This work presents some good concepts and definitions on acquiring data through SQL-based mechanism but it is limited on relational databases and has less support to context data management.

PerLa [19] is a SQL-like language designed for collecting data from different nodes in a pervasive system. PerLa can acquire data from different sources independently from the underlying structure by dividing queries into two levels. However, PerLa does not adequately consider context data processing but limits sensor data. CML [12] queries context data directly through SQL by converting its context data model which can represent different type of context data to relational database schema through its internal designed facility. However, this conversion may generate complex SQL join operations and query representations are not programming and platform independent.

Another SQL-based context querying mechanism is described by Judd and Steenkiste [8]. By defining four types of entities (i.e., device, access point, people and space), this framework utilizes a Context Service Interface-SQL (CSI-SQL) wrapper to query and access data from context sources, but the primary function of this design is on expressing attribute requirements, timely execution and meta-attribution rather than context data. Fjords architecture [10] designs some SQL-based facilities to manage and query over streaming sensor data with supporting both pull- and push-based data acquisition methods. The mechanism presented by Madden et al. [11] can also acquire sensor data either through pull queries or push queries as well as a hybrid queries.

By studying the existing SQL-based CQLs, we observe that no one has fully fulfilled the requirements we illustrated in Section II. Motivated by this, we propose and design this SQL-based CQL leveraging on these previous works.

B. Non-SQL based CQL

RDF-based CQL is the other most powerful CQL in context-aware computing as shown in [6]. A typical RDF-based CQL is the MUSIC CQL proposed by Reichle et al. [17] which has a good support on querying derived information. However, supporting only single entity severely limits its querying capability. Another RDF-based querying mechanism described by Perich et al. [14] decouples queries into sub-queries and process them separately to overcome some obstacles posed by mobile environment. However, this reliance on mobile devices creates availability and reliability issues. SOCAM [5] can easily provide context information about entities and relationships between entities in the smart space leveraging on the ontology technology, but it has limitations in supporting complex queries.

Although RDF-based CQLs are more suitable for expressing relationships of context sources and are as powerful as SQL-based CQLs, they are not as flexible as SQL-based CQLs and produce large amounts of overhead in creating classes, sub-classes and relationships [14]. Furthermore, a large percentage of existing systems are using SQL-based methods to manage and acquire data, which are not easy to integrate with RDF-based approach. As a result, we plan to devise this SQL-based language so that the learning curve is not steep and integration is faster.

IV. PROPOSED CONTEXT QUERY LANGUAGE

Motivated by the need of a CQL that can fulfill all the requirements discussed previously, we propose a new SQL-based CQL to express context queries. As the language is designed leveraging on SQL syntax, the query structure consists of the existing SQL constructs like SELECT… FROM… WHERE with GROUP BY…HAVING…ORDER BY…as optional instructions. Most of the existing CQLs just extend SQL with basic SELECT…FROM…WHERE structure without mentioning other optional instructions to support querying on context data. In this proposed query language, we add some constructs to the existing SQL query structure to support them. These constructs include MODE, SUBSCRIBE/UNSUBSCRIBE, ON VALUE…LIFETIME which will be discussed in detail and which syntax is given in this section. The underlying context model leverages on the concept of context domain to divide and manage different categories of context sources, and each context domain is represented by a list of context attributes to represent context data, just as shown by by Pung et al. [16]. The basic structure of context query is as follows with constructs that are enclosed by […] being optional:

MODE GLOBAL | LOCAL
SELECT | SUBSCRIBE | UNSUBSCRIBE
<attribute> | <contextEvent> | <operation (attribute)> | <operation (contextEvent)> [, attribute, …]
FROM <domain> [, domain, …]
[ON INTERVAL <interval> **LIFETIME** <timespan>**]**
WHERE <predicate> [**AND** | **OR** predicate …]
[GROUP BY <attribute**>]**
[HAVING <predicate**>]**
[ORDER BY <attribute**>] [DESC | ASC]**

A. Description

The **MODE** clause specifies the mode of the query which is a new construct added specially for this CQL. Nowadays, with the advances of ubiquitous computing, data about various kinds of environmental conditions and other entities can be obtained and utilized by applications. On the other hand, with the advances of sensing technology on mobile devices, many applications which only utilize context information of a host device have also been designed and implemented. We believe that a properly designed CQL should be able to support these two types of applications effectively. By analyzing the characteristics, we can see that processing context queries of local applications should be different from global applications. In order to address this issue, we divide context queries into two types and use the MODE construct in the proposed CQL with possible values LOCAL and GLOBAL respectively. LOCAL represents queries of applications which utilize context information

 ISBN: 978-1-61208-364-3

from their host devices only; while GLOBAL indicates queries of applications which demand context data from different context sources. As a result, this proposed CQL supports both application types by supporting their corresponding data acquisition conditions.

The **SELECT/SUBSCRIBE/UNSUBSCRIBE** clause specifies the required context information or actions. The SELECT construct is inherited directly from SQL and represents queries that acquiring context data in a pull-based manner. However, since context-aware applications aim to detect changes of situations and adapt to them automatically, there are many queries concerning the changes of context information. We call these context changes as context events which will be pushed to applications as notifications when they are triggered. Unfortunately, the pull-based SELECT construct cannot support those push-based kind of queries. In order to address this issue, we design this new **SUBSCRIBE** construct which lets the application developers issue push-based queries. We also design the **UNSUBSCRIBE** construct to let applications cease their reception of notifications about certain kinds of context events.

In addition, we extend the **SELECT/SUBSCRIBE /UNSUBSCRIBE** clause with three different types of expressions to indicate what type of context information is required by the query, namely: context attribute, context event, and operation involved context. *Context attribute* is inherited from SQL but extended to indicate context information of a specific context attribute of a specific context domain. In the case of traditional database, data is integrated, these attributes can also be relational attributes.

Besides the simple context attribute, we also define two new types of context information: context event and operations involving context. The *context event* represents context information about changes of context source status and triggering notifications. As discussed previously, context event is an important type of context information for context-aware applications. Autonomy of context-aware applications is realized through situation detection and event notification mechanisms. By tracking the changes of situational status, those context events can trigger corresponding notification mechanism to make applications adapt to new situation automatically. It takes the format like "*domain. contextEvent*" which includes three sections: domain name, delimiter(.) and event name. The same event name may appear in different context domains, so the context domain information is shown to solve the possible ambiguity.

The other newly defined type of context information is the *operation involved context* which indicates context information derived by applying certain context processing operations on raw context data. Unlike traditional database systems which mainly focus on data updating and retrieval, context-aware systems also need to interpret or derive higher level context information during the query answering process. In order to generate the higher level context information, appropriate functions should be applied on collected raw context data. Even though traditional database systems also provide certain aggregating functions, they are not powerful enough to handle other contextual based operations. In order to solve this problem, we propose this new type of context information to represent those operation involved contexts. The expression consists of two parts: *operation* and *context data*. Operation indicates what context processing operations are utilized, while context data specifies raw context data required for the operations. This context data can be either context attribute or context event as illustrated above. There are several types of operations that can be applied on context data and we provide a separate section to give more details in next sub section.

```
<context> ::= <context attribute> |
   <context event>  | <operational
   context>
<operational context> ::= <operations>
   (<context attribute> | <context
   event>)
<context attribute> ::= <context
   domain>.< attribute>
<context event> ::= <context domain>.<
   event>
<context domain> ::= PERSON |OFFICE |
   HOME | SHOP | CLINIC | CAR ...
<attribute> ::= name | location |
   temperature | mood | activity |
   humidity | luminanicity | ...
<event> ::= locationChange | moodIsSad
   | temperatureIsHigh | ...
<operation> ::= <aggregating function>
   | <algebraic function> |
   <contextual function>
```

The **FROM** clause is also inherited directly from SQL, but we extend it to specify the context domains involved in the context queries. Those context domains are predefined by context-aware systems. In the context of Coalition [16], which organizes context sources into different domains, the values can be PERSON, OFFICE, HOME, SHOP, CLINIC, etc. Additionally, these context domains can be easily extended to relations of traditional databases. As a result, traditional database can be easily integrated with context data to produce higher order information. In the future implementation, it can also be extended to include semantic web sources or other Internet sources.

```
<context domain> ::= PERSON |OFFICE |
     HOME | SHOP | CLINIC | CAR |...
```

The **ON INTERVAL … LIFETIME** clause is a new construct designed for supporting context queries about continuous or periodic context acquisition. Just as discussed in Section II, context can be continuous data or periodic data. There are many context-aware applications that need to acquire context data continuously. One typical example is monitoring the security status of a critical place. Additionally, periodic acquisition of context information is also common for context-aware applications and a typical example is to monitor the patient status in hospitals. However, traditional database systems utilizing SQL which usually focus on handling sporadic data retrieval cannot

 ISBN: 978-1-61208-364-3

represent and handle these continuous or periodic cases appropriately. In order to solve this problem, we design this new construct and augment it to the proposed CQL.

The **ON INTERVAL** clause of the construct specifies the sampling interval for a context query. In the design, we treat continuous context retrieval as a special case of periodic context retrieval with the interval as zero. In order to differentiate this special case, we use a special value NULL to represent the continuous nature of retrieval. On the other hand, we use normal integer value plus time unit (i.e., ms, s, min, h) to indicate periodic context queries. Irrespective of whether the context retrieval is continuous or periodic, the usual pattern is that context information is retrieved over a period of time. In order to represent this issue, we design the **LIFETIME** clause of the construct to indicate the timespan of the context retrieval. There are two extreme cases of this timespan values. One is zero which implicitly indicates that the query is neither a continuous or periodic query. The other one is everlasting which means that the retrieval will never stop. Even these two cases are rare, but we include them for completeness and represent them with NULL and EVER respectively. A normal timespan value is represented by an integer value with a corresponding time unit.

The **WHERE** construct is inherited directly from SQL but extended to represent the list of constraints on context information acquisition. Constraints are the heart of a query since they provide a guideline on how to filter out unwanted context data from the large number of available context data sources. Most of existing SQL-based CQLs utilize simple predicates only, which is restrictive in expressing complex requests. In this CQL, compound predicates that consists of more context constraints connected by AND/OR are built.

```
<compound predicate> ::= <disjunctive
   predicate> AND <disjunctive predicate>
   AND ...
<disjunctive predicate> ::= <simple
   predicate> OR <simple predicate> OR ...
<simple predicate> ::= <context
   expression> <operator> <context
   expression>
<context expression> ::= <context
attribute> | <context event> | <context
constant> | <functional expression>
```

The remaining three constructs **GROUP BY, HAVING** and **ORDER BY** are all directly inherited from SQL, but extended with context variables or predicates as the arguments. Most of the previous SQL-based CQLs do not extend SQL with those constructs. However, we think these constructs are also necessary, especially in querying context data from a large number of context sources, in which case we may need these construct to further process the final results. The **GROUP BY** clause defines how the resulting context information can be further grouped with respect to specific context information. The **HAVING** clause defines how the filtered context data can be further selected with respect to certain conditions represented by having-predicates in the definition. The **ORDER BY** clause defines how the query results should be sorted with respect to the attribute indicated by context information either in ascending order (**ASC**) or descending order (**DESC**). These three constructs are all optional.

B. Context Processing Functions

Context processing functions represents various kinds of operations that can be applied on context data to generate or derive higher level context information. In addition to the updating and reading operations done by traditional database system, context query processing also takes charge of interpreting context data and deriving higher level information, which is just realized by those context processing functions. Some of the existing CQLs have similar processing operations integrated like [1] [14] [17].

SQL currently provides five aggregating functions that can be applied on a set of data to get some insights to the data patterns and behavior, namely: MAX, MIN, SUM, AVG, and COUNT. We think these aggregating functions are also necessary for context data query processing and inherited directly from SQL.

```
<aggregating function> ::= SUM |
   COUNT | AVG | MIN | MAX.
```

Another important type of functions added for the CQL is *contextual functions* which provide the task of context information interpretation and higher level information derivations from basic context data. Traditional database systems usually do not provide any functions for data interpretation. However, this type of functionalities is very important for CQL, so an important aspect of the proposed CQL is to support contextual functions that can be used by applications for information interpretation and reasoning. It is important to provide a suite of different types of such functions to support a wide variety of applications. One set of such contextual functions can interpret the basic sensor data and draw inferences based on the data and certain predefined logical conditions. For instance, isFever(temperature) interprets whether a given temperature corresponds to a fever or not. Similarly, an isFire(temperature) function can be used to determine a fire in a building and generate appropriate functions. Another class of contextual functions provides the task of deriving relations between two or more entities. For instance, isFriend(personA, personB) checks whether two persons are friends. This function can utilize the social network and contact information of a person to make this decision. Another function nearBy(x, y) can compute whether two entities are within a certain distance from each other. There are also contextual functions aiming to derive situational information. For instance, isMeeting(location) will determine whether there is a meeting going on in the given place. These are just some of the examples of the different types of contextual functions that we propose to support as part of our context query language.

```
<contextual function> ::= isFever(t) |
 distance(a, b) | isFriend(a, b) |
```

 ISBN: 978-1-61208-364-3

```
nearby(x, y) | isFire(a) |
isMeeting(l) | ...
```

Since these contextual functions are usually situation and application dependent, it is not possible to predefine and generate an exhaustive list of all the contextual functions. Instead, the application developers can define functions according to their application requirements, which requires the CQL to be extensible with new contextual functions. In order to realize this extensibility, we propose the creation of a contextual function repository that can hold the different types of contextual functions as defined by application developers. Additionally, this approach also promotes reusability as the popular contextual functions can be shared among applications. During the query processing, whenever a contextual function is detected, the query processor can retrieve the function definition and related rules from this repository to analyze the collected data and generate result accordingly.

V. EVALUATION

We implement this proposed CQL with our context-aware system Coalition [16] and examine the querying processing performance. In this section, we focus on illustrating how the proposed CQL can represent different types of requests and fulfill those requirements presented in section II and illustrate the proposed context query language usage with a study case.

A. Discussions

The proposed CQL expresses context domains, context attributes and conditions in a conceptual level. In other words, the expressions focuses on expressing what a user wants and does not mention any details about how context sources and context data are managed in the underlying framework or system. This means the proposed CQL express context should be able to express context queries at an abstract level.

Also, the proposed CQL utilizes a generic and conceptual method to model context data wherein context sources are divided into different context domains and each domain is associated with a list of context attributes. This mechanism is consistent with the concepts of relation and attribute in traditional relational database.

The proposed CQL supports two types of queries to support context acquisition: select-based queries and subscribe-based queries, which correspond to pull-based or push-based information retrieval methods respectively. Select-based queries specify the required context information with a list of conditions to filter out unwanted context data. This type of queries is issued on demand and gets context data in real time. On the other hand, subscribe-based queries issue required context information with a list of constraints proactively and the context data will be pushed to the query issuer whenever the constraints are triggered.

Additionally, the proposed CQL defines different functions to process context data to generate higher level context information. The aggregating functions can provide some pre-processing operations on the context data, while contextual functions can apply some predefined rules on context data to derive situational information.

The design of ON INTERVAL with LIFETIME section enables the proposed CQL to design context queries for continuous or periodic data retrieval. Additionally, the complex structure design of the WHERE clause enables the proposed CQL to express complex constraints and requirements. Together with specifying different domains in the FROM clause, the proposed CQL can represent compound queries that retrieve and process context data from different domains and context sources.

Another advantage of this proposed CQL is that it can be integrated with a traditional relational database system. As the proposed CQL is actually an extension of SQL, it can easily involve relational data by replacing context domains with relations. At the abstract level, application developers actually have no idea whether the data comes from context sources or relational database. As a result, context data and relational data can be seamlessly integrated. From the above illustration and analysis, we can see that proposed CQL has fulfilled the requirements illustrated in section II.

B. Case Study

In this sub section, we are going to illustrate the usage of this proposed context query language with a shopping guide scenario.

Lisa went shopping in shopping mall X during last weekend. At the moment of entering the mall, Lisa received a message about new arrivals from one clothes shop A as she subscribed the notification service previously. After going around at shop A, Lisa received a recommendation for the bookstore shop B based on her preference of book and current location. During the time in shop B, Lisa realized her friend Emily was also shopping in the same mall, and then she contacted Emily for a coffee. By checking the real time queuing information of the several coffee shops, they chose shop C, after which they looked around together and had a good shopping time.

In this scenario, different types of context information need to be queried. First, in order to push notification of new arrivals, the context event of Lisa's entering the shopping mall X should be subscribed in advance. Secondly, in order to give proper recommendations to Lisa, we need to find shops based on both her hobby and location context. Thirdly, in order to check whether there are friends around, we need to provide the function of find nearby friends based on current location. Last, in order to provide the real time queuing information, we need to monitor the queues of each shop. Following four context queries are designed for these four points respectively:

Q1: (***Context Event Subscription***)

```
MODE          GLOBAL
SUBSCRIB      person.enterMall("Mall X")
FROM          person
WHERE         person.name = "Lisa"
ON INTERVAL   0
LIFETIME      EVER
```

Q2: (***Search Matched Shops)***

```
MODE      GLOBAL
SELECT    shop.name
FROM      shop, person
WHERE     shop.type = person.preference
    AND   isNearby(shop.location, person.location)
    AND   person.name = "Lisa"
```

Q3: (***Check Nearby Friends***)

```
MODE      GLOBAL
SELECT    person.name
FROM      person
WHERE     isNearby(person.location, "Mall X")
    AND   isFriend(person.name, "Lisa")
```

Q4: (***Query Queuing Information***)

```
MODE      GLOBAL
SELECT    shop.name, MIN(shop.queue)
FROM      shop
WHERE     shop.type = "coffee"
    AND   person.location = "Mall X"
```

VI. CONCLUSION AND FUTURE WORK

We presented a new SQL-based CQL in this paper. By exploring the properties of context data and pervasive environments, we illustrated the requirements for a well-designed CQL. Through studying existing CQLs, we observed that SQL-based CQLs are more flexible and easy to utilize, inspired by which, a new SQL-based CQL is proposed and designed. This CQL supports both pull- and push-based queries as well as continuous context retrieval by different time intervals. Additionally, different context processing functions, especially contextual functions, have been designed to generate higher-level context information. Furthermore, this CQL supports compound conditions to get context data from various context entities belonging to different context domains. Leveraging on the proposed SQL-based CQL, user requests can be better represented and supported by the underlying context data management system. One of the main future works is to further integrate context reasoning operations with this proposed CQL in which reasoning operations can be represented by certain intermediate context processing operations, so that referenced context information can be generated in runtime during context query processing.

ACKNOWLEDGMENT

This research was carried out at the SeSaMe Centre. It is supported by the Singapore NRF under its IRC@SG Funding Initiative and administered by the IDMPO.

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". Human-Computer Interaction, 2001, pp. 97-166.

[2] L. Feng, "Supporting context-aware database querying in an ambient intelligent environment". In: 3rd IEEE International Conference on Ubi-media Computing (U-Media), July 2010, pp. 161-166.

[3] L. Feng, J. Deng, Z. Song, and W. Xue, "A logic based context query language". Smart Sensing and Context, 2010, pp. 122-134.

[4] C. Fra, M. Valla, and N. Paspallis, (2011). "High level context query processing: an experience report". In: IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), March 2011, pp. 421-426.

[5] Gu, T., Pung, H. K., and Zhang, D. Q. A service-oriented middleware for building context-aware services. Journal of Network and Computer Applications, 2005, pp. 1-18.

[6] P. D. Haghighi, A. Zaslavsky, and S. Krishnaswamy, "An evaluation of query languages for context-aware computing". In: 17th International Conference on Database and Expert Systems Applications (DEXA), 2006, pp. 455-462.

[7] J. Heer, A. Newberger, C. Beckmann, and J. I. Hong, "Liquid: context-aware distributed queries". In: Ubiquitous Computing (UbiComp 2003), October 2003, pp. 140-148.

[8] G. Judd and P. Steenkiste, "Providing contextual information to pervasive computing applications". In: First IEEE International Conference on Pervasive Computing and Communications (PerCom 2003), March 2003, pp. 133-142.

[9] Y. Li, L. Feng, and L. Zhou, "Context-aware database querying: recent progress and challenges". The Book Context-Aware Mobile and Ubiquitous Computing for Enhanced Usability, 2009, pp. 147-168.

[10] S. Madden and M. J. Franklin, "Fjording the stream: an architecture for queries over streaming sensor data". In: 18th International Conference on Data Engineering, March 2002, pp. 555-566.

[11] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: an acquisitional query processing system for sensor networks". ACM Transactions on Database Systems (TODS), 2005, pp. 122-173.

[12] T. McFadden, T., Henricksen, K., and Indulska, J. "Automating context-aware application development". In: UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management, Nottingham, September 2004, pp. 90-95.

[13] F. Perich, A. Joshi, T. Finin, and Y. Yesha, "On data management in pervasive computing environments". IEEE Transactions on Knowledge and Data Engineering, 2004, pp. 621-634.

[14] F. Perich, A. Joshi, Y. Yesha, and T. Finin, "Collaborative joins in a pervasive computing environment". The International Journal on Very Large Data Bases, 2005, pp. 182-196.

[15] M. Perttunen, J. Riekki, and O. Lassila, "Context representation and reasoning in pervasive computing: a review". International Journal of Multimedia and Ubiquitous Engineering, 2009, pp. 1-28.

[16] H. K. Pung, et al. "Context-aware middleware, for pervasive elderly homecare". IEEE Journal on Selected Areas in Communications, 2009, pp. 510-524.

[17] R. Reichle, et al. "A context query language for pervasive computing environments". In: 6th Annual IEEE International Conference on Pervasive Computing and Communications. March 2008, pp. 434-440.

[18] O. Riva anD C. di Flora, "Contory: a smart phone middleware supporting multiple context provisioning strategies". In: 26th IEEE International Conference on Distributed Computing Systems Workshops, July 2006, pp. 4-7.

[19] F. A. Schreiber, R. Camplani, M. Fortunato, M. Marelli, and G. Rota, "Perla: A language and middleware architecture for data management and integration in pervasive information systems". IEEE Transactions on Software Engineering, 2012, pp. 478-496.

 ISBN: 978-1-61208-364-3