# Context Processing: A Distributed Approach

Penghe Chen
*NUS Graduate School for Integrative Sciences and Engineering*
*National University of Singapore (NUS)*
*Singapore*
*g0901858@nus.edu.sg*

Shubhabrata Sen
School of Computing (SOC)
National University of Singapore (NUS)
Singapore
g0701139@nus.edu.sg

Hung Keng Pung
School of Computing (SOC)
National University of Singapore (NUS)
Singapore
dcsphk@nus.edu.sg

Wai Choong Wong
Department of Electrical & Computer Engineering
National University of Singapore (NUS)
Singapore
elewwcl@nus.edu.sg

***Abstract*—Context processing refers to the operation of processing different types of context data and/or information using different kinds of operators. These operators are applied according to some conditions or constraints given in context queries. Existing context aware systems process context data in a centralized fashion to answer context queries and generate context information. However, this method can cause scalability issues and give poor system throughput. In this paper, we aim to address this issue by proposing a distributed context data processing mechanism in which the context data processing computations of different context queries will be distributed to different computing devices. Relying on the developed prototype, a performance evaluation was conducted with centralized context data processing method as benchmark.**

***Keywords*-Context; context-aware; ubiquitous computing; data management; distributed; context processing; query plan**

## I. Introduction

The advances in the field of ubiquitous computing have resulted in the proliferation of context aware applications that are flexible, adaptable and capable of acting autonomously on behalf of users (1). Context is any information that can be used to characterize the situation of an entity (2), and context awareness dictates the state in which devices or software programs are aware of the situation and adapt changes automatically, without requiring explicit user intervention (1). In order to truly realize context awareness, various kinds of context aware systems are developed to utilize context information about the situation of its users, the environment, or the state of the system itself to adapt their behavior (3).

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. Among these different operations of context data management, context processing is the process in which different context data and/or information is processed with different kinds of operators or operations applied according to some conditions or constraints given in context queries. Context processing is the operation where lower level data is converted to higher level context information that is more meaningful and useful for users, so it is very important for a well developed context aware system. A centralized context data processing approach is usually adopted due to its simplicity in development and management.

However, as the demand for context information grows with the advent of advanced context aware applications a centralized data processing approach may not be adequate. One of the issues is that the central point will become a bottleneck when handling large scale context queries which will cause a scalability issue. Also, it severely affects the system throughput which we define as the number of context queries handled in a unit time interval.

Motivated by this, we proposed a distributed context processing mechanism through which context processing computations of different context tasks will be distributed and executed on different computing devices automatically. As a result, system performance upon system throughput and resource utilization can be improved. The contribution of this paper can be highlighted in two aspects. Firstly, we propose a way to build context processing plan automatically in runtime, and divide context processing operations into several independent parts. Secondly, leveraging on the context processing plan, we propose a way to process context data for different tasks automatically in a distributed manner.

The rest of the paper is organized as follows. A system overview of our context aware system Coalition is given in Section 2. Section 3 will illustrate the details of distributed context processing mechanism and its work flow. Performance evaluation is demonstrated in Section 4. Section 5 discusses the related work and the whole paper is concluded in Section 6.

## II. System Overview

Coalition (Fig. 1) is a four layer based context aware system: physical operating space layer, data management layer, service management layer and application layer. The physical operating space layer includes all physical entities such as sensors, actuators and computing devices that provide the actual context data. The context data management layer aims to provide effective and efficient context data organization, lookup and storage. The service management layer manages context-aware services to handle the consolidated applications. Application layer supplies interfaces that are used to invoke services or acquire data from lower layers. The work of this paper focuses on the data management layer.

Coalition utilizes two main concepts to manage context sources - physical space and context space. A physical space is a person, object or place and provides a software module for managing context data and communicating with other components named as physical space gateway (PSG). Context spaces are defined as domains of physical spaces with similar context attributes, and the software module is context space gateway (CSG). Leveraging on these two concepts, Coalition could supports SQL-based context queries through the query processor component. More details of Coalition can be found in (4).

## III. Distributed Context Processing

As discussed in the introduction, context processing is the process by which context data and/or information is processed using different kinds of operators or operations applied according to some conditions or constraints given in context queries. The previous version of Coalition utilizes a centralized context processing method to handle context queries and process context data. However, this can lead to a scalability issue in handling large number of context queries and affect system throughput. The proposed distributed context processing mechanism will handle these issues through decoupling the computations of context processing of queries from the other operations. As a result, context data processing of different queries can be executed independently and in parallel to improve system scalability and throughput.

In this section, we first discuss the basic concepts related to a context query language. We then proceed to discuss the details of the context processing operation and introduce the notion of a query plan. Finally, we illustrate how these concepts can be utilized to decouple the context processing computation from other operations in order to realize a distributed context processing mechanism.

### A. Context Query Language

A context query language (CQL) is the language with specific syntax used by a context aware system to query context information. Context query language is crucial for querying

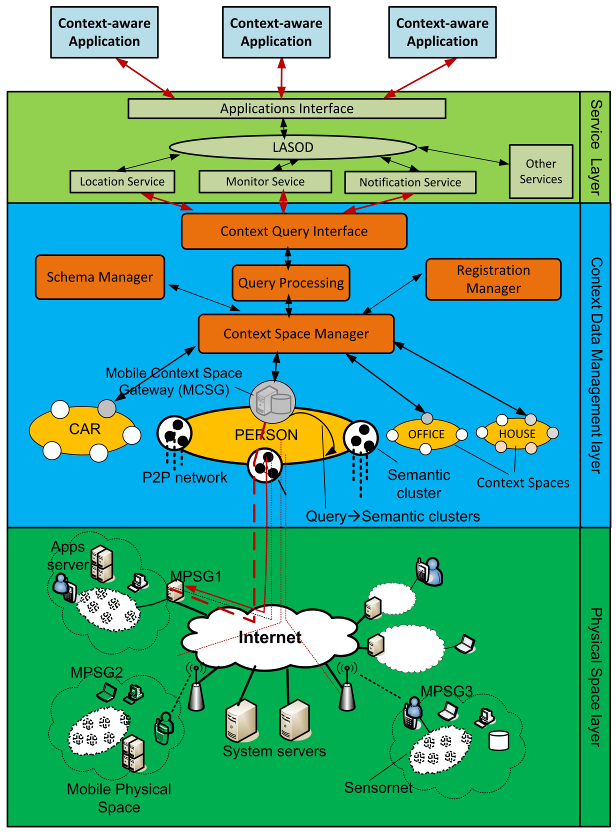

Figure 1. Coalition Architecture

context as it defines the way of query represented and context information required (5). Also, a good CQL should consider heterogeneity of context sources and different types of operations like reasoning, filtering and aggregating (6).

Many different types of context query languages have been proposed and designed in previous work, such as RDF-based, XML-based, SQL-based and Graph-based CQL. An evaluation work done by (5) indicates that SQL -based and RDF -based query languages are better than others. In our work, we will leverage on SQL-based Query Language to represent context queries. The basic structure of context query is as following:

**SELECT** (context attribute)
**FROM** (context domain)
[**WHERE** (constraints)]

The design of this CQL is inspired by normal SQL, so the syntax and format is quite similar to normal SQL, and that is why we call it SQL-based CQL. Due to the specialties of context data management, we have to admit that, comparing with normal SQL, this CQL has less query handling capability. This CQL cannot handle complex queries containing more context attributes and context domains, and it does not support the functionalities of ”HAVE” and

”SORT BY”. Additionally, the actual query analysis and processing operations are different from normal relational database management system. Other than the key words, the semantics of the queries are restricted by the concepts inside this context aware management system. A typical example of context query in our system likes – SELECT preference FROM PERSON WHERE location = ”Shop A”.

## B. Context Processing Operation

An important aspect of context processing is the context processing operation (CPO) which represents the specific methods that transform low level context data into high level context information. For example, an aggregation operation can represent a summarized view of several low level context data items. Context processing operations are closely related to context query language. Based on the CQL defined in previous subsection, we can outline the different types of context processing operations as follows

- **Filtering** - operations that aim to retrieve a piece or a set of context information leveraging on certain constraints to filter out unnecessary context information.
- **Aggregating** - operations that mainly include functions of SUM, AVG, COUNT, MAX, and MIN of a set of context information, and it is utilized together with filtering operations.
- **Reasoning** - operations that aim to generate higher level context information by applying certain user defined rules or methods on a set of context data.
- **Matching** - operations that aim to match two or more different pieces of context information based on certain constraints.
- **Sorting** - operations that aim to sort a set of context information in certain order based on certain context information or certain constraints.
- **Merging** - operations that aim to consolidate a set of context information together based on certain constraints.

## C. Query Plan

Based on the notions of CQL and CPOs discussed previously, we propose the concept of query plan which aims to provide a generic representation of context query information. Query plan is defined as an object that contains a list of attributes to represent context query information and a list of methods for retrieving these attribute values. As per the current design, the query plan contains the information about the query issuer, context domains, context attributes, context processing operations and the query constraints. An important piece of information is the details of the query issuer that represents the address of the PSG that issues a context query. The inclusion of this piece of information in the query plan lets the PSGs holding the relevant answers reply to the query issuer directly instead of going through the query processor as an intermediary. As a result, the context processing operations can now be carried out independently. This query plan is generated from parsing the query according to the CQL and CQOs defined previously. Any subsequent operations in query processing can easily get the query information directly from the query plan object without parsing the original query. As a result, the query information becomes independent from CQL. We utilize this observation to develop a decentralized approach to perform the context processing operation. A diagrammatic illustration of query plan is shown in Fig. 2.

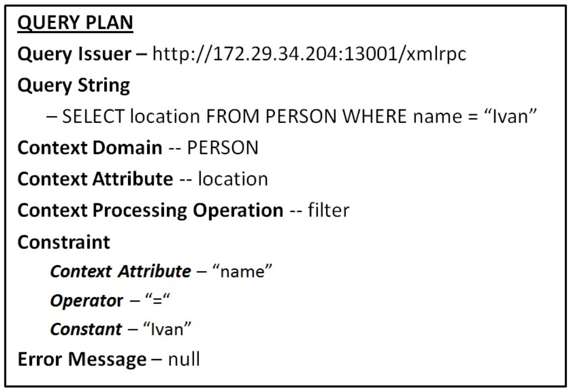
**QUERY PLAN**
**Query Issuer –** http://172.29.34.204:13001/xmlrpc
**Query String**
– SELECT location FROM PERSON WHERE name = “Ivan”
**Context Domain** -- PERSON
**Context Attribute** -- location
**Context Processing Operation** -- filter
**Constraint**
***Context Attribute*** – “name”
***Operato*r** – “=“
***Constant*** – “Ivan”
**Error Message** – null

Figure 2. Example of Query Plan

## D. Details of Previous Query Processing Operation

Based on the previous discussion, we observe that there is a possibility of implementing a distributed context processing mechanism to solve the scalability issue and improve system throughput. The previous query processing operation utilized in the middleware can be roughly divided into the following phases as follows:

***Query Parsing***: parse the context query based on specified CQL syntax and extract corresponding information of context query. This phase processes context queries at syntax level.

***Query Analysis***: based on the information obtained during query parsing, further analysis is applied to extract more information, like required context information, context domain involved and constraints of the query. This phase s interprets the context query.

***Query Distribution***: based on information extract in query analysis phase, we further identify the context sources involved in each context query and distribute the query to the relevant PSGs to collect the necessary context data.

***PSG Context Data Collection***: after receiving the query, each PSG will parse the query and check whether its context data satisfies the constraints stated in the query and report the result to the query processor.

***Context Processing***: consolidate the required context information by applying the context processing operation extracted from the query in the analysis phase on context

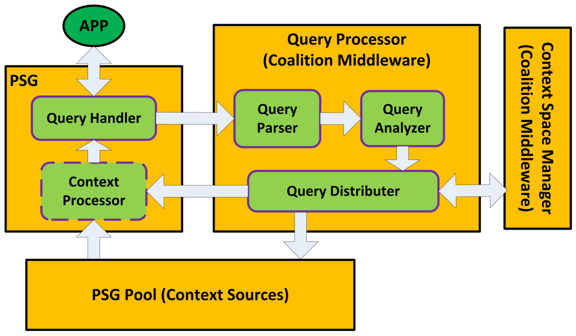
APP
PSG
Query Handler
Context Processor
Query Processor (Coalition Middleware)
Query Parser
Query Analyzer
Query Distributer
Context Space Manager (Coalition Middleware)
PSG Pool (Context Sources)

Figure 3. Architecture of Distributed Context Processing

data collected from different PSGs. This work is done in query processor of Coalition middleware, and all queries utilize this single context processor to generate required context information.

In the subsequent sections, we illustrate how the proposed distributed processing approach modifies and improves the aforementioned query processing operation to realize better scalability and throughput performance.

## E. Distributed Context Data Processing

The new architecture design of query processing based on proposed distributed context processing mechanism is shown in Fig. 3. Different from previous design, in this architecture, Context Processor resides in PSG side rather than the query processor in Coalition middleware. Additionally, instead of utilizing a single context processor for all context queries, this new design creates an individual context processor for each PSG. As a result, those context processors can work simultaneously for different context queries to improve system performance.

The design of the distributed context processing mechanism includes two tasks. A major task is to decouple the context processing phase from the other phases. The next one is to shift and distribute the context processing computations of different context queries among different PSGs. These two tasks are solved leveraging on the design of query plan which enables the five phases of query processing to be loosely coupled by recording query parsing results. The other phases can retrieve any query information directly from this query plan object. As a result, the PSG context data processing phase does not need to re-parse the query and can execute independently from other phases.

Query plan can also help to distribute computations of context data processing into different PSGs. In the centralized context processing mechanism, all data is processed in the query processor where each PSG reports a matching query answer to. However, in the proposed distributed context processing mechanism, the PSGs need to be informed the place where to report their context data and a PSG processing the context data needs to know the relevant operations to be applied on the data. These functionalities can be provided using the query plan. The PSGs can utilize the query issuer information to identify where the data needs to be sent. Also, the PSGs can utilize the context processing operation information to identify the operations to be applied on the data.

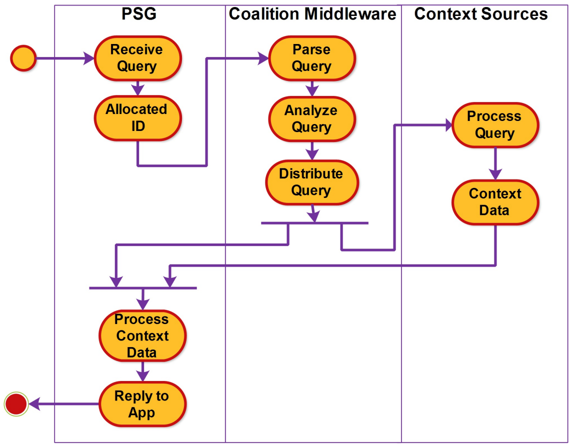

Figure 4. Work Flow Diagram

## F. Work Flow of Distributed Context Processing Mechanism

In order to better demonstrate how this distributed context processing mechanism handles context queries, we describe the operational work flow in this subsection. A diagrammatical illustration of the work flow leveraging on UML activity diagram is given in Fig. 4.

At the beginning, an application issues a context query to its host PSG which will generate a unique ID for the query. Also, PSG forwards the query to the query processor in Coalition middleware. After receiving the query, the query parser of query processor will generate a query plan for this query by parsing it based on SQL-base CQL syntax. Also, this query parser will check the validity of context domain and context attribute information wrapped inside the query. Leveraging on the query plan object, query analyzer will extract out information of context domain and context attribute, and then identify the group of PSGs of context sources involved. Subsequently, query distributer retrieves a random PSG reference from Coalition middleware utilizing on extracted context domain and context attribute information. Query distributer forwards query plan to host PSG to create context processor. Meanwhile, it also forwards query plan to the random PSG which will then flood query plan to all PSGs involved leveraging on P2P network. Query handler residing on each PSG of context source will check their context data fulfills the requirements stated in query plan and reports to corresponding context processor if it is. Context processor created in host PSG applies context processing operation stated in query plan on all the collected context

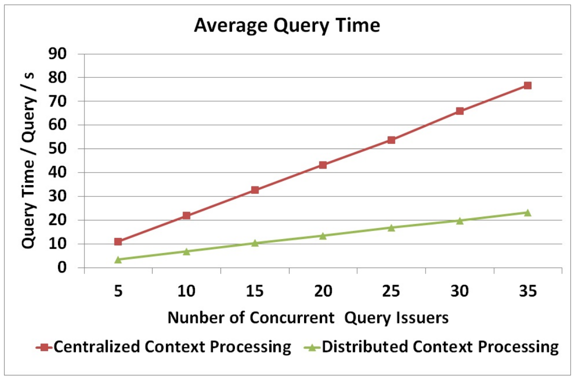

Figure 5. Average Query Time

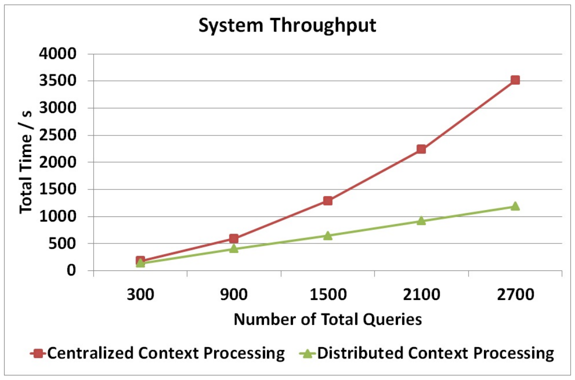

Figure 6. System Throughput

data to generate the required context information and then reply the context information to application.

## IV. Performance Evaluations

We have implemented a prototype of this distributed context processing mechanism and integrated with our context aware system Coalition (4). A Dell PowerEdge T300 server with four 2.83 Ghz quad-core Intel Xeon CPU was used as the server. PSG instances are simulated by desktop PCs where each PC had an Intel Core 2 Duo 2.83 GHz CPU and runs the Windows XP OS.

Several hundreds of PSGs from five different context domains were simulated: PERON, HOME, OFFICE, SHOP and CLINIC. Queries of different domains and different context information were continuously issued to test the average query time and system throughput. A range cluster based mechanism is used to restrain the number of PSGs to be flooded for each query (7). The centralized context processing method utilized by previous version of Coalition is used as a benchmark to evaluate the performance of this proposed distributed context processing algorithm.

### A. *Average Query Time*

We first studied the average query time in case of different number of users issuing queries concurrently. The query time is measured as the interval between the issuing of a query and the reception of the context information. Average query time is measured by letting all users issue 100 queries simultaneously and continuously. The measurements of the currently utilized centralized context processing method are utilized as a benchmark. The results are illustrated in Fig. 5. The results indicate that our proposed distributed context processing provides a significant improvement on the average query time. The average query time is an important indicator of system scalability because a system can easily been break down if average query time increase so rapidly to overtake the responding time threshold with the number of user increasing. By observing the results, we can say that our proposed mechanism makes the system more scalable now.

### B. *System Throughput*

Another performance indicator studied is system throughput which measures the maximum number of queries that can be processed within a unit time interval. Since counting the maximum number of queries handled is not easy especially for a distributed system, we measure the time consumed for certain number of queries. Methods with shorter total responding time means they can handle more queries in a given time period and have a higher throughput. Fig. 6 illustrates the total response time in case of different number of total queries issued with the condition that 30 users concurrently issue queries. Compared with centralized method, our proposed distributed context processing method requires shorter time to process the same number of context queries. This reflects that our proposed distributed context processing method can increase the throughput of the middleware with regards to the number of queries processes within a certain time period.

### C. *Query Time Breakdown*

We also analyzed the time breakdown for the operations included in context query processing namely the following - query preprocessing, query distribution and context processing. The query preprocessing event refers to the operations involved in query processing including query parsing and query analyzing. The query distribution process refers to the flooding of the query to PSGs and local query processing in each PSG. This also includes the queuing time for issuing queries and reporting context data to query processor. The context processing operation refers to the task of processing context data to generate context information. Due to space constraints, we only present one example here in which 20 users issue queries concurrently and continuously. Table 1 illustrates time spent of each operation in query processing. We observe that the query preprocessing and context processing time does not change too much but query distribution

TABLE I. QUERY TIME BREAKDOWN

| | Centralized Context Processing (/s) | Distributed Context Processing (/s) |
|---|---|---|
| **Query Preprocessing** | 2.3 | 2.4 |
| **Query Distribution** | 38.9 | 8.8 |
| **Context Processing** | 2.1 | 2.2 |

is significantly improved. This is consistent with the design of distributed context processing mechanism which relives the work load and communication congestion of Coalition middleware. As a result, the PSGs do not need to wait longer to issue queries and report context data.

## V. RELATED WORD

There is little previous work in context aware computing that explores specifically on how to process context data. Most of existing work focuses on either single domain with centralized context processing or multiple domains but centralized context processing. This state of the art demonstrates the proposal of this distributed context processing algorithm is novel and promising.

Some of existing systems managing context data in single space or domain utilize centralized context processing method. CoBra (8), which is an agent based context aware system in smart spaces, utilizes a specialized component called Context Broker as the central point to reason and store context information on behalf of applications. Semantic Space (9) leverages on semantic web technologies to manage context data in smart spaces. Context aggregator gathers various context markups from different context sources and reports to context knowledge base which takes charge of answering context queries, but both context aggregator and context knowledge are centralized. The heart of CMF (10) is a centralized black-board based context manager which collects context data from all participating services or notes and processes those context data to generate proposer context information for querying. Trimmed to mobile devices, Hydrogen (11) leverages on various adaptors to collection context data. Meanwhile, it replies a centralized Context Server in management layer stores all the context information to handle context retrieval and subscription.

Some systems manage context data in multiple spaces or domains, but utilize centralized context processing method. SOCAM (12) can manage context data from different domains and is efficient in reasoning higher level context information leveraging ontology based context modeling technique. However, the data management and reasoning relies on pulling all context data into a central semantic space. CASS (13) utilizes a server based mechanism to abstract higher level context information and support SQL based queries. The server serves as a central point to collect various context data from sensor notes and derive higher context information leveraging on rule engines. Toolkit (14) designs various types of widgets to manage context data. Context aggregators and context interpreters are built to aggregate context data and derive context information. These aggregators and interpreters are distributed and executed for different purpose, but it lacks a systematic interface to handle context query but let the developers specify all required context sources in higher level. Solar (15) utilizes operators to produce various kinds of context information, in which computations are done in a distributed manner like our mechanism. However, Solar requires application developers manually define and create those operators, while our mechanism can automatically generate the processing operators and produce the final context information in runtime. As a result, application developers will not be decoupled from details of lower level context data management.

## VI. CONCLUSION AND FUTURE WORK

We propose a distributed context processing mechanism in this paper that aims to handle large scale context queries. Leveraging on the design of query plan, we decouple context processing from the other query processing operations. As a result, the proposed mechanism can execute context processing part in separate computing devices; thereby improving the system scalability and throughput. Additionally, a performance evaluation is done by taking previous centralized context processing method as benchmark. The experimental results demonstrate that the proposed technique performs better in terms of system scalability and throughput. As part of our future work, we plan to extend the query capability in handling more complex context queries which may include more complicated context processing operations with different conditions and more context data from different context domains.

## REFERENCES

[1] M. Baldauf, S. Dustdar, and F. Rosenberg, “A survey on context-aware systems,” *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, 2007.

[2] A. Dey, “Understanding and using context,” *Personal and ubiquitous computing*, vol. 5, no. 1, pp. 4–7, 2001.

[3] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, “A survey of context modelling and reasoning techniques,” *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161–180, 2010.

[4] H. Pung, T. Gu, W. Xue, P. Palmes, J. Zhu, W. Ng, C. Tang, and N. Chung, “Context-aware middleware for pervasive elderly homecare,” *Selected Areas in Communications, IEEE Journal on*, vol. 27, no. 4, pp. 510–524, 2009.

[5] P. 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*, 2006, pp. 455–462.

[6] R. Reichle, M. Wagner, M. Khan, K. Geihs, M. Valla, C. Fra, N. Paspallis, and G. Papadopoulos, “A context query language for pervasive computing environments,” in *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, ser. PERCOM ’08, 2008, pp. 434–440.

[7] S. Sen and H. Pung, “A mean-variance based index for dynamic context data lookup,” *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pp. 101–112, 2012.

[8] H. Chen, T. Finin, and A. Joshi, “An intelligent broker for context-aware systems,” Ph.D. dissertation, 2003.

[9] X. Wang, J. Dong, C. Chin, S. Hettiarachchi, and D. Zhang, “Semantic space: an infrastructure for smart spaces,” *Pervasive Computing, IEEE*, vol. 3, no. 3, pp. 32–39, 2004.

[10] P. Korpipaa, J. Mantyjarvi, J. Kela, H. Keranen, and E. Malm, “Managing context information in mobile devices,” *Pervasive Computing, IEEE*, vol. 2, no. 3, pp. 42–51, 2003.

[11] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, and W. Retschitzegger, “Context-awareness on mobile devices - the hydrogen approach,” in *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS’03)*, 2003.

[12] T. Gu, H. Pung, and D. Zhang, “A service oriented middleware for building context aware services,” *Journal of Network and Computer Applications*, vol. 28, no. 1, pp. 1–18, 2005.

[13] P. Fahy and S. Clarke, “Cass - middleware for mobile context-aware applications,” in *Workshop on Context Awareness, MobiSys 2004*, 2004.

[14] A. Dey, G. Abowd, and D. Salber, “A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications,” *Human-Computer Interaction*, vol. 16, no. 2-4, pp. 97–166, 2001.

[15] G. Chen, M. Li, and D. Kotz, “Data-centric middleware for context-aware pervasive computing,” *Pervasive and Mobile Computing*, vol. 4, no. 2, pp. 216–253, 2008.