Title	A semantic context management framework on mobile device
Author(s)	Hu, DH; Dong, F; Wang, CL
Citation	The 6th IEEE International Conference on Embedded Software and Systems (ICESS 2009), Zhejiang, China, 25-27 May 2009. In Proceedings of the 6th ICESS, 2009, p. 331-338
Issued Date	2009
URL	http://hdl.handle.net/10722/125706
Rights	International Conference on Embedded Software and Systems. Copyright © IEEE.

A Semantic Context Management Framework on Mobile Device *

Dexter H. Hu, Fan Dong, Cho-Li Wang
Department of Computer Science
The University of Hong Kong
Pokfulam Road, Hong Kong
{hyhu, fdong, clwang}@cs.hku.hk

Abstract

We present a semantic context management framework named ContextTorrent*, which can make various types of context information be semantically searchable and sharable among local and remote context-aware applications. We implement this framework on the Google Android platform with its elegant application support. An open source RDF parser has been extended to effectively get RDF triples from files or over the network. Three embedded database systems were evaluated for storing ontology represented contexts in the resource-constrained mobile devices. We use the FOAF ontology schema and a synthetic data set of up to 2500 records to evaluate the context query and storage performance. Ordinary context queries can be replied instantaneously.*

1 Introduction

Pervasive and ubiquitous computing promise to transcend traditional desktop computers to physical smart spaces. It envisions a world where people can always access the right information at the right time anywhere. A key component that realizes this vision is *context-awareness*, an application's ability to detect and react to environment variables during its interaction with the ambient environment [15].

Many context-aware applications have been engineered recently. Most of these efforts focus on middleware support to fast prototype context-aware applications within local smart spaces [13]. These middlewares provide commonly reusable software components to develop, integrate, and deploy context-aware applications [16, 14, 18]. However, most of these systems were built for specific application domain and could be used only by a small user group. Real-life pervasive applications that benefit people's daily life are still far from fruition. There are several reasons. First, there lacks standardized infrastructure to provide real-life context information about public utilities, such as traffic condition, weather forecast, community notice, bus schedule, etc. Although these context information are generally available in public, semantic interoperability among multiple types of context sources remains a challenging issue. Second, personal context information can be of many different types and natures. A typical context-aware application usually has to make use of both high-level contexts (e.g., email addresses, to-do list, and SMS/IM messages) and low-level contexts (e.g., GPS location information, network condition, and physical device configuration). Moreover, large volume of spatial-temporal contextual data have to be handled as a city inhabitant's daily activities may take place across multiple smart spaces and span among various duration. Third, the explosive growth in the popularity of mobile devices and the social networking software have made the management of context data more challenging due to the frequent location-dependent interactions and more complex social relationships. The interactions among different levels of context data and their spatial-temporal linkage ought to be thoroughly exploited to realize large-scale context-aware applications that benefit people's daily life.

The whole process of context acquisition, interpretation and adaptation [24] calls for an abstract and unified context management framework to facilitate context searching, sharing, and transferring. The Semantic Web (OWL/RDF) [19] provides a common framework that allows data to be shared and reused across applications and community boundaries. The linkage of data in Semantic Web forms a giant global graph, which has an intuitive analogy to the semantic relationships among context entities. These useful semantic relationships can be exploited to power the context-aware applications, and make them more intelligent. The potential of using Semantic Web for cross-domain knowledge representation has also made it a

*This work is supported in part by HKU Basic Research programme (Grant No. 10400018) and National Natural Science Foundation of China (Grant No. 60773089).

978-0-7695-3678-1/09 $25.00 © 2009 IEEE
DOI 10.1109/ICESS.2009.95

promising technology for effective context sharing among context-aware applications with different nature, particulary for those demanding frequent context update during a long standing process.

We introduce *ContextTorrent*, a generic yet efficient context data management framework which leverages the Semantic Web technology. *ContextTorrent* could make various types of context information be semantically searchable and sharable among local and remote context-aware applications. In this framework, a mobile device could act as a context provider or a context consumer, or both at the same time. These small devices could be connected to their desktop counterparts, devices in a smart space, and other nearby mobile devices through an overlay peer-to-peer network. To achieve fast local and remote context query and provisioning, we first use the ontology-based context modeler that consistently manage semantic relationships among RDF triples, which are dynamically generated or modified. These relationships are further persistently reflected to an embedded object-oriented database, which consumes a relatively small storage overhead on resource-constrained mobile devices. Lastly, due to the semantic relationship linker, multiple concurrent applications can get their desired contexts quickly with much better perceived experience. The proposed framework has the following new features:

- *Unified Context Representation*. We treat all types of context equally regardless if they are low-level or high-level contexts, or inherit different spatial-temporal characteristics.
- *Lightweight Semantic Labeling of Native and Extensible Context Sources*. Existing context information and extensible context source are labeled or connected as semantic resources. Additional context types can be easily defined with minimal programming effort through Android's Content Provider [3] interface.
- *Embedded Object-Oriented Database for Storing Semantic Relationships*. We adopted an embedded object-oriented database scheme to store the rapidly changing semantic relationships with little storage space overhead, while achieving fast query response.
- *Adaptive Relevant Context Suggestion*. We record context-aware application's interaction with contexts on mobile devices, such that this information will have indirect enforcement for future relevant context search, when dealing with semantic relationship ambiguity. Related context entities will be ranked by the N-gram based matching algorithm.

The prototype of *ContextTorrent* was built atop of the Google Android platform [2]. We demonstrate the programmability and simplicity of using *ContextTorrent* by implementing a context-aware *personal information management* (PIM) application. We took advantages of Android rich API to uniformly represent the low-level system data (e.g. WiFi, GPS data) and high-level context (e.g., user profile). This application can automatically create a contact entry for each new incoming call, and let user associate other semantic resources as this contact's property values, which are further materialized to persistent storage on mobile devices. The optimized VM instance process of Android, together with its Activity [1] switching, have greatly improved user perceived experience in sharing real-life context information on mobile platforms.

The rest of this paper is organized as follows. We review related work in Section 2. Our proposed design is explained in Section 3. We report the experiment settings and evaluate query and storage performance in Section 4. Section 5 concludes the paper with future work.

2 Related Work

Our work is inspired from an independent research field *semantic desktop*, which combines semantic technologies with traditional desktop search for personal information management (PIM).

The *semantic desktop* is a unified data abstraction layer which supplements the user's long-term memory by automatic provisioning of related information based on contextual relevance. Typical functionalities of a semantic desktop include filing and finding related documents in a desktop computer. The Gnowsis system [23] is a good example of semantic desktop systems. It has a back-end daemon performs data processing and provides storage service for native applications. External application that wants to utilize the metadata can be easily integrated with standardized interfaces. However both Gnowsis and its successor NEPOMUK are rather heavy-weight, for example, the HTTP interface of typical RDF frameworks (Jena and Sesame) does not directly fit in a normal mobile device.

"Stuff I've seen" (SIS) [17] is another similar project that supports unified index of contents for email, web page, document, appointment, etc. It provides an interface for rich contextual information discovery. Naturally, we extend these unified semantic data management services to the *context* management framework for context-aware applications, particularly in mobile situation that calls for light-weight query/storage, semantic inter-interoperability and easy application development.

Although there already exist many popular PIM softwares [11, 5] on desktop or mobile devices, their usage scenarios are simply for archiving or browsing. There are two main problems in existing PIM systems. First, these PIM softwares lack a common data management framework, which can address the semantic aspect of data consumption

in a networked environment. This is further complicated in the pervasive computing environment, which consists of heterogeneous software platforms and device capabilities. Second, the personal information consumption goes beyond simple data synchronization or sharing. Despite some standard integration protocols (e.g., SyncML [12]) and synchronization platforms (e.g., Sun Open Mobile Sync [9]), legacy software vendors can not benefit from the promising Semantic Web technologies.

WikiCity [22] provides a digital representation of a real-world city that allows city inhabitants to access, add, and modify information and enjoy location-based services, including nearest emergency facilities, information broadcast on public spaces. Core of WikiCity is a real-time data exchange platform that accommodates various types of public utility information from heterogeneous sources. Manhattan Story Mashup [26] is another interesting large-scale pervasive system that combines the Web, camera phones, and a large public display. Its storytelling involves a new form of interaction, which lets distant people collaborate in real-time. However the underlying mechanisms for connecting photo and story content are simple. We believe the high-level semantic relationships could be exploited to achieve more advanced features in storytelling. Story Mashup also lacks a formal infrastructure for massive data delivery, when the number of game participants become large.

Live Mesh [8] and DropBox [4] are more recent industry softwares that could be used to synchronize a group of shared files among peers. However the data sharing and synchronization in Live Mesh are limited to ordinary user files. How to manage context information containing semantic relationships among entities was not addressed. *ContextTorrent* supports fine-grained synchronization control on shared data. Through a more flexible context entity labeling scheme, more accurate and relevant context data can be searched and located.

3 ContextTorrent Architecture

3.1 Overview

Figure 1 shows an overview of the proposed *ContextTorrent* framework and the interactions among its core components.

We use standard Semantic Web interface (e.g., OWL and RDF) for context modeling, as in many pervasive applications [20]. The *ontology-based context modeler* (OCM) models various types of contexts as semantic resource using an embedded RDF/OWL parser that parses external RDF files or RDF triples brought in by the *overlay peer networking* component over the network. Besides parsing the OWL/RDF schema, the context modeler also performs insertion, deletion and modification of RDF triples, either obtained from from the network or imposed by the local *semantic relationship linker*, to dynamically update semantic relationships.

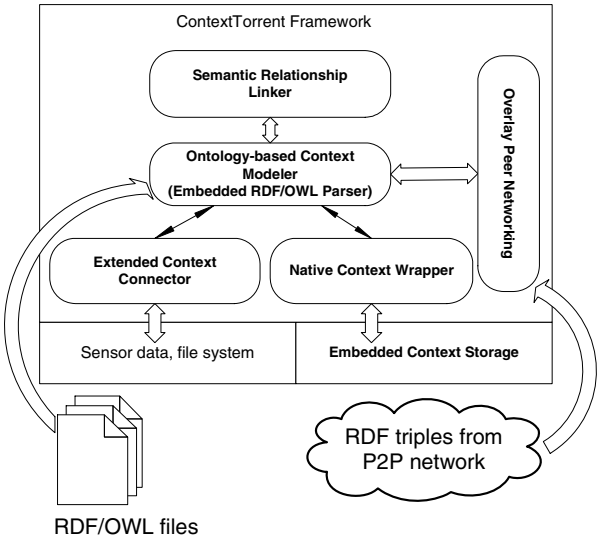

Figure 1. The Overview of ContextTorrent Framework

The *semantic relationship linker* (SRL) is responsible for the management of semantic relationships among context entities. Any context-aware application can issue a context query to SRL, which will return a context resource URI according to its relevancy ranking. This context resource URI could later be used to locate all its associated properties to get the property values.

To provide unified interface for accessing context data, *native context wrapper* (NCW) and *extended context connector* (ECE) are developed. NCW wraps up existing and application-defined context data using ontology schemas, making them accessible by applications via standard URIs. ECE creates links to external context sources (e.g., file resources on a remote device) and does the format adaptation of various sensor data (e.g. GPS receiver). Unique URIs are created for all context sources. The separation of access interface and data storage management makes it possible for the upper layer context modeler to build customized and extensible storage independently. For example, we build the *embedded context storage* (ECS) based on an embedded object-oriented database to store the ontological relationships among context entities.

Lastly, the *overlay peer networking* (OPN) component is implemented to transfer RDF triples among nearby mobile devices, their desktop counterparts, or other devices in a smart space. We adopt a peer-to-peer overlay network to connect all these devices. Such overlay peer-to-peer

networking could facilitate context searching, provisioning, and delegation more efficiently in a large-scale network environment.

The rest of this section explains the technical details of SRL, OCM, and ECS, three main components that are crucial for the realization of *ContextTorrent* framework.

3.2 Semantic Relationship Linker

Relationships between resources are defined as *properties* in OWL. Besides manual creation of rather static semantic relationships between resources, in reality more useful properties are created during user's interaction with context-aware applications. An appropriate mechanism to capture these runtime contexts and dynamically extend the semantic relationships could improve the context awareness of the applications.

The *semantic relationship linker* is designed to capture application-level interactions at runtime and link up semantic relationships in a dynamic fashion. The key functions of SRL are realized based on the concept of SmartLink [1], originally used to associate words in a plain text with more meaningful semantic data. For example, we can link a presentation file (e.g. ppt, pdf) to an "Activity" of a "Calendar" event.

We make use of Android's Intent [7] interface to enable context sharing across applications. It is an abstraction that combines an operation with the data to operate on. This feature enables the runtime binding of program code of concerned applications to certain shared context data, thus efficient multitasking can be achieved, which is a key advantage over other Java-based mobile platform (e.g. J2ME). So context sharing across applications will have much better user experience. We also make use of the *notification* mechanism of Android to implement context-aware notifications.

SRL is also responsible for maintaining a persistent storage to reflect the usage statistics, through native context wrapper to update the ECS. Since all context-ware applications will send its context query to *semantic relationship linker* by simple keywords or RDF triples with variables, *semantic relationship linker* can easily record various usage statistics of context entity URIs. We feed the instance data into native storage by Android's Content Provider [3]. Every context-aware application could issue its context query to SRL, which will return a matched context URI to further retrieve its associated property values.

Besides the above basic functions, the following enhancements are provided in SRL to improve the quality of context search: (1) We adopt an *N-gram algorithm* [21], a widely used string matching algorithm in sequence analysis, to do fuzzy search on related resource entities. We extend the *CentralTagging* of OpenIntent [2] to label context resources. It enables client applications to edit tags that are related to a context entity. This method is helpful to improve the similarity ranking of the basic N-gram matching algorithm. (2) We design a *chained relationship tracker*, which involves two or more depth traverse of existing relationships to identify more related entities. (3) For each property of a context entity, we attach a *degree rank* to reflect its importance, which is recomputed whenever it is selected for use. The definition of the *degree rank* can be as simple as its usage frequency, or subject to any relevancy update made by other applications. (4) We allow off-line annotation of relationships on newly generated context from environment. For example, we record each new incoming call as a new contact entry and let user fill other detailed information.

3.3 Ontology-based Context Modeler

The *ontology-based context modeler* (OCM) models both *static* and *dynamic* contexts. Static contexts are generally those high-level information about personal particulars, description of a resource entity, etc. These information are contained in RDF/OWL files that can be imported at start time. We adopt published ontologies, such as PIMO [10] and Friend Of A Friend [6], to model personal information and social relationships. User can also extend existing ontology schema to define new relationships as OWL properties. All these static contexts are stored in *embedded context storage*.

Dynamic contexts are mainly those spatial-temporal information (e.g. GPS trajectory of the mobile user), which are used by applications to trigger actions. To support time-based and location-based context modeling and provisioning, OCM annotates every event sample in a form of RDF triple (*subject, predicate, object*). For example, when an RFID reader detects an object, an RDF triple (*subject, predicate, object*) = (*CT-subject:readerID, CT-predicate:Detect, CT-object:ObjectID*) will be created to update local semantic relationships or trigger the adaption of a local application. The RDF triple could also be transferred to a remote peer by the *overlay peer networking* component to activate necessary actions in a remote application.

To import static contexts and update dynamic contexts, OCM has implemented an embedded RDF/OWL parser which can feed in contexts as RDF triples from files or over the network. We port an open source NanoXML RDF parser [3] from J2ME platform to Android platform to realize part of the functions required by OCM. The new parser can quickly traverse relationships among resource entities to extract their properties and values.

[1] http://code.google.com/p/smartlink/

[2] http://code.google.com/p/openintents/

[3] http://nanoxml-j2me.wiki.sourceforge.net/

The OCM also does the insertion, deletion and modification of RDF triples that are imposed by semantic relationship linker, in order to consistently maintain semantic relationships. Figure 2 shows the typical functions to parse and traverse RDF resources.

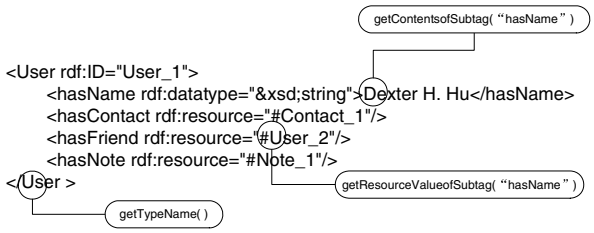

Figure 2. Functions supported to manipulate materialized OWL by NanoXML

This component is also responsible for translating semantic context queries (e.g. RQL) to native queries that can be handled by NCW and ECC. The semantic context queries usually come from a remote peer's context subscription.

3.4 Embedded Context Storage

To store ontology represented context information on mobile devices, we need an efficient persistent storage scheme. The semantic relationships among different context entities resemble links in a connected graph. While traditional relational database management systems (RDMS) pose as big flat files, it is not an intuitive reflection of semantic relationships.

The *embedded context storage* (ECS) provides persistent storage services for *ContextTorrent*, which is implemented on an Object-oriented Database Management System (ODBMS). We propose to adopt an object-oriented approach to store context information as the object-oriented concept has a good analogy to the ontological representation. The object-oriented database allows users to define arbitrary types of context as objects without much burdening the program designers (e.g. troubles in using RDMS schema) or administration of databases. Therefore, programmers are able to focus more on the business logic of context-aware applications. Moreover, the *schema evolution* mechanism of object-orient database could be leveraged to dynamically reflect relationship changes to persistent storage.

Although ODBMS provides good programmability, it potentially could consume more storage space and be less efficient in context search, which are essential in a mobile context-aware application. This might require specifically optimized indices on different data types to speed up a context search. We investigated two different implementations of object-oriented database: *Perst* [4] vs. *db4objects*[5], and compared their query performance against the built-in SQLite database.

We use Content Provider of Android to implement *native context wrapper* (NCW), which allows the context modeler to connect to *embedded context storage* (ECS), regardless which database model is used. Existing context data that are pre-stored in the built-in database can also be accessed via the native context wrapper.

4 Evaluation

4.1 Experiment Setting

We implemented *ContextTorrent* in Android 1.0 SDK emulator, Release 2 on a Linux machine with 2GB RAM and an Intel Duo core CPU running at 2.66GHz. The emulator configures any application with maximum memory size of 16MB.

Existing context data, such as “contacts”, “call log”, and “bookmarks” could be pre-stored in the native storage place and accessed via the native context wrapper. We also define additional context sources that represent “task/to-do list”, “email”, “IM message”, “calendar event”, “GPS location history”, and “buddylist”.

At current stage, we adopt the JXTA [25] P2P protocol, and have ported the JXME (JXTA on J2ME framework) as a background service in Android to transfer context RDF triples (e.g. GPS trace) over the network.

We prototyped a context-aware PIM management application to demonstrate the use of various components implemented by *ContextTorrent* that facilitate context searching, sharing, and transferring on mobile devices. For example, we are able to issue a query to find a person's phone number with his name as the keyword. The returned result is a FOAF URI, which is used later to navigate its associated property values. We adopted the FOAF ontology schema and prepared a synthetic data set of up to 2500 records to evaluate the query and storage performance. The NanoXML RDF parser feeds the FOAF data sets into Android's built-in SQLite, db4o (version 7.4.84) and Perst (version 4.02) separately.

4.2 Query & Storage Performance

Since the user experience on mobile device is critical for its adoption, we evaluate the query latency when a user tries to find a related context entity.

The user experienced query latency can be divided into two parts: (1) *link retrieval*: time to retrieve all semantic

[4] http://www.mcobject.com/perst/
[5] http://www.db4o.com/

links to be ranked later on; (2) *fuzzy search*: time spent on similarity matching among retrieved semantic links to find the most relevant context entity. We also evaluate the time of popping up a context entity's properties, denoted as *property popping*, after the user determines the context entity to navigate.

Figure 3 compares the consumed storage space (in byte) with respect to the number of context entities stored in the database. As we can see, the storage consumption of Perst is around 2-3 times larger than the built-in SQLite RDMS. The db4o is the poorest, which can consume up to 10 times more space than SQLite. So among object-oriented databases, Perst is better than db4o in terms of space requirement.

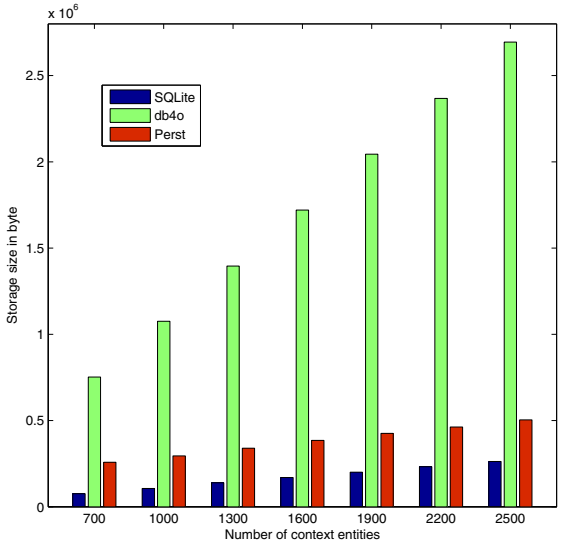

Figure 3. Storage consumption comparison

Figure 4 shows the continuous query time of the three database schemes. The tested data set consists of 2500 context entities. We report the total query time by increasing the number of continuous queries. The Perst database performs the best with the fastest query time, and the slowest scaling slope. The db4o remains the worst among the three. These results can be explained by Figure 5, which shows the average latency time for randomly querying a single context record with respect to different database sizes (i.e., number of context entities). We take the average latency of 20 random context queries for each database size and plot the performance curves. As shown, the query time for all databases are relatively stable under various data sizes. The query time of db4o is the largest, while the query time for Perst is the smallest, and is much faster than SQLite.

The reason behind is that Perst database has optimal indices for different primitive data types, including R-Tree (for GIS data), T-Tree (for in-memory access), Patricia Trie,

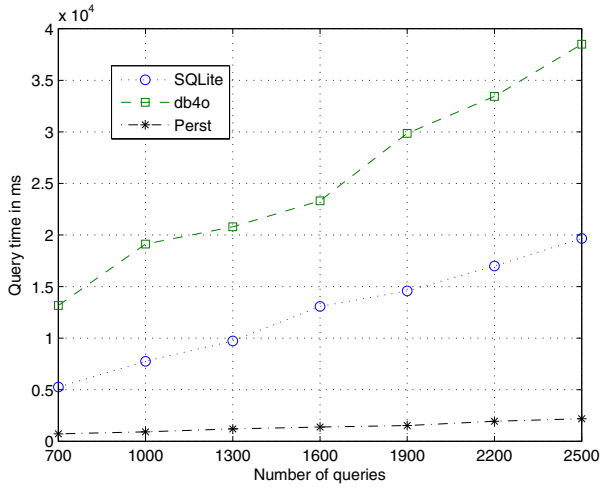

Figure 4. Continuous query test

and many others. The dedicated index type is especially important for mobile devices with limited memory, and scarce CPU/storage. With the above comparisons, we conclude that Perst is the best database scheme among the three, because it has the fastest query speed and a relatively small storage overhead than the built-in SQLite and db4o.

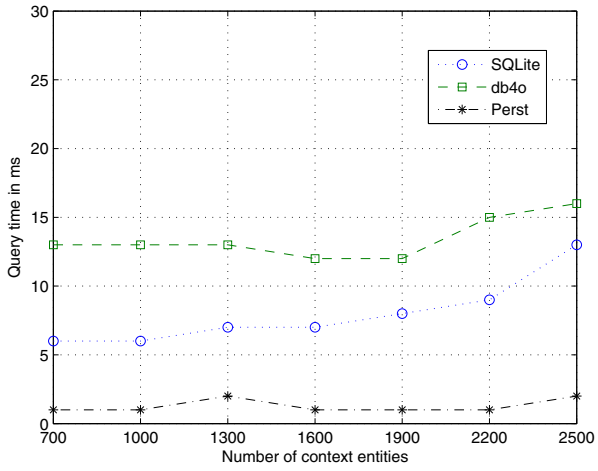

Figure 5. Evaluation on the query response time w.r.t database size

Figure 6 shows the latency breakdown with respect to *link retrieval* and *fuzzy search* time under three database schemes. Test cases with response time exceeding 3000 ms were truncated because three seconds are the normal latency tolerance of user experience. As we can see, Perst is the best among the three, while db4o is the worst. Note that the

link retrieval time of SQLite is almost negligible (5-10 ms). This is because SQLite does not directly return a set of context entities, but a cursor that can iterate through the context entities. For db4o, the total query time exceeds three seconds when context entity size is 1000 or more. Thus db4o is the worst scheme among the three.

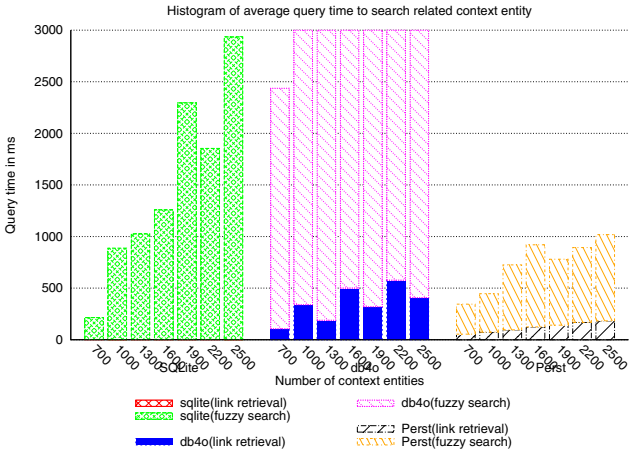

Figure 6. Breakdown of user experienced latency under various data size

Despite the linear growth of query time as the data size increases, we found the N-gram similarity matching algorithm could consume more than 80% of the total latency time in all cases. This could be a performance bottleneck, when there exists more context types and large context volume. We will improve this fuzzy search on the Perst embedded database by tag filtering or imposing time-bound query in our future work.

We also took a closer look at the *property popping* time for returning different number of properties upon a query of a single context entity. The time consumed involves activity creation and switching in Android, and the sorting on the *degree rank* (usage frequency in this case) on different properties for the requested context entity. Figure 7 shows that the time used for popping up properties is nearly instantaneous (less than 100 ms), when the number of returned properties is less than 30. This result is satisfactory because an ordinary user is unlikely to navigate a large number of properties about the resource entities in a single query.

5 Conclusions

In this research, we address various issues of context data management and provisioning for supporting context-aware computing. The proposed *ContextTorrent* framework can semantically organize, search, and store various types of contexts and their semantic relationships using an ontology-

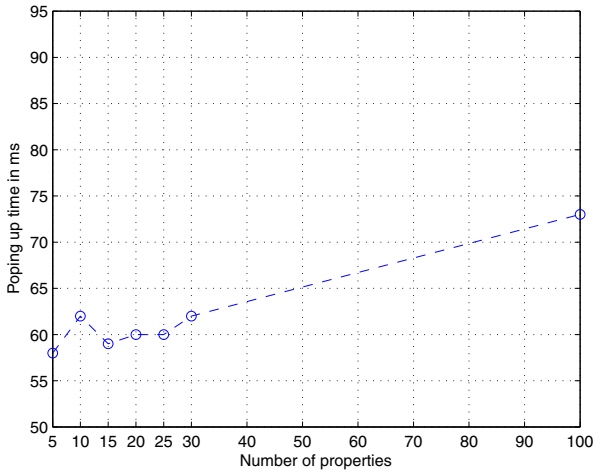

Figure 7. Time consumed by popping up properties of a context entity

based context modeler. We leverage several powerful features of Google's Android platform to make context sharing among multiple applications efficient and lightweight. In general, given any OWL/RDFs schema and its associated instances, it is possible to build a tool within Android to automatically parse the schema with Intent declaration for all subjects' properties. It can also feed the instance data into native storage by Content Provider, such that other applications can easily access these semantic resource entities by their URIs.

Through intensive evaluation, we show that context queries using the API of *ContextTorrent* could perform efficiently. The proposed context annotation scheme using RDF triple could be a promising solution to support advanced context provisioning in a large-scale network. We also evaluate three database schemes for managing context storage on mobile devices. The Perst embedded database turns out to be the best as compared to SQLite and db4o. The object-oriented database allows users to define arbitrary types of context as objects, which seems to be the right choice for context storage.

Future work includes the design and implementation of a fully-fledged desktop *ContextTorrent* counterpart (e.g., based on KDE-Nepomuk), which is compatible with the mobile version to form a *context provisioning network*. An important task in this design is the implementation of QoS control to cope with both system-level performance metrics (e.g., delivery latency, data quality) and high-level context-aware application requirements.

References

[1] Activity: http://developer.android.com/reference/android/app/activity.html.

[2] Android platform: http://developer.android.com/.

[3] Content provider: http://d.android.com/guide/topics/providers/content-providers.html.

[4] Dropbox: http://www.getdropbox.com/.

[5] Essentialpim: http://www.essentialpim.com/.

[6] Foaf vocabulary specification: http://xmlns.com/foaf/0.1/.

[7] Intent: http://developer.android.com/reference/android/content/intent.html.

[8] Live mesh: https://www.mesh.com/welcome/default.aspx.

[9] Open data sync: http://dsc.sun.com/learning/javaoneonline/2008/pdf/ts-5957.pdf.

[10] Personal information model ontology: http://dev.nepomuk.semanticdesktop.org/wiki/pimoontology.

[11] Pimone software: http://www.pimone.com/pimone.htm.

[12] The syncml initiative: http://www.openmobilealliance.org/.

[13] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277, 2007.

[14] G. Biegel and V. Cahill. A framework for developing mobile, context-aware applications. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications*, pages 361–365, March 2004.

[15] A. K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, February 2001.

[16] A. K. Dey, D. Salber, and G. D. Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2–4):97–166, 2001.

[17] S. Dumais, E. Cutrell, J. Cadiz, G. Jancke, R. Sarin, and D. C. Robbins. Stuff i've seen: a system for personal information retrieval and re-use. In *Proceedings of the 26th ACM Annual International Conference on Research and Development in Informaion Retrieval*, pages 72–79, 2003.

[18] S. Long, R. Kooper, G. D. Abowd, and C. G. Atkeson. Rapid prototyping of mobile context-aware applications: the cyberguide case study. In *Proceedings of the 2nd Annual International Conference on Mobile Computing and Networking*, pages 97–107, 1996.

[19] S. Lu, M. Dong, and F. Fotouhi. The semantic web: opportunities and challenges for next-generation web applications. *International Journal of Information Research*, 7(4), 2002.

[20] R. Masuoka, Y. Labrou, B. Parsia, and E. Sirin. Ontology-enabled pervasive computing applications. *Intelligent Systems, IEEE*, 18(5):68–72, 2003.

[21] E. Miller, D. Shen, J. Liu, and C. Nicholas. Performance and scalability of a large-scale n-gram based information retrieval system. *Journal of Digital Information*, 1(5), 2000.

[22] B. Resch, F. Calabrese, A. Biderman, and C. Ratti. An approach towards real-time data exchange platform system architecture. In *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, pages 153–159, 2008.

[23] L. Sauermann, G. A. Grimnes, M. Kiesel, C. Fluit, H. Maus, D. Heim, D. Nadeem, B. Horak, and A. Dengel. *Semantic Desktop 2.0: The Gnowsis Experience*. 2006.

[24] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Proceedings of Workshop on Mobile Computing Systems and Applications*, pages 85–90, Dec 1994.

[25] B. Traversat, M. Abdelaziz, D. Doolin, M. Duigou, J. C. Hugly, and E. Pouyoul. Project jxta-c: enabling a web of things. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, pages 282–290, 2003.

[26] V. Tuulos, J. Scheible, and H. Nyholm. Combining web, mobile phones and public displays in large-scale: Manhattan story mashup. In *Proceedings of the Fifth International Conference on Pervasive Computing*, pages 37–54, 2007.