

Caching Context Information in Pervasive Systems

Mylone Anandarajah and Jadwiga

Indulska*

School of ITEE

The University of Queensland
Brisbane, QLD 4072, Australia

{mylone, jaga}@itee.uq.edu.au

Ricky Robinson

Queensland Research Laboratory

National ICT Australia Limited

300 Adelaide Street, Brisbane, QLD 4000,
Australia

ricky.robinson@nicta.com.au

ABSTRACT

Context aware systems are systems that use context information to adapt their behaviour or the content they provide. This paper addresses the problem of disconnections between nodes in these systems. Disconnections in a context aware system may occur because of node mobility, network failures or node failures. A research opportunity lies in improving the robustness of such systems to disconnections. While traditional distributed systems methods of improving robustness in the face of disconnections can be applied to context aware systems, the additional metadata available to them may be leveraged to provide smarter caching algorithms. This paper presents ideas for context information caching algorithms that utilise such metadata.

Categories and Subject Descriptors

H.3.4 [Systems and Software]: Distributed systems

Keywords

context-awareness, pervasive computing, caching

1. INTRODUCTION

1.1 Context Aware Systems

Context is information about the situation of a person, place or object [2]. When we as humans interact with other people we adapt our behaviour depending on the physical and verbal signals we receive from them. For example if the person we are interacting with is showing signs of sorrow we will interact with them in a certain way; if the person is thrilled we will interact with them in a different way. These signals provide us with context. It informs us on how to approach a particular interaction. Computers, however,

*The authors are also affiliated with the Queensland Research Laboratory, National ICT Australia Limited, Brisbane, QLD 4000, Australia.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MDS '06, November 27-December 1, 2006, Melbourne, Australia.
Copyright 2006 ACM 1-59593-418-9/06/11 ...\$5.00.

are not as good as humans in using context to adapt their behaviour to suit the situation they find themselves in [9]. Much research is being carried out to determine how applications can effectively use context information.

Context aware systems are systems that use context information to adapt their behaviour or the content they provide. A context aware system consists of three main components: Context Sources; Context Sinks or Context aware Applications; and Context Management Systems.

Context sources produce context information of the following types [7]: Sensed data which is produced by sensors; user defined (profiled) data which is entered into the context aware system by the user; static data which is data that does not change; derived data is data which is inferred from other data. To give an example of derived data, a context aware system may infer that a meeting is taking place when there are several people within a room that is known to be used for meetings and the door is closed. Each type of data has its own characteristics. For example sensed data is prone to noise.

This classification of context information as sensed, static, user-defined or derived can be viewed as metadata: that is, information about context information. Other kinds of context metadata include the freshness and update frequency of context information. The presence of this context metadata means applications can make better informed adaptation decisions.

The context management system is a middleware layer between the context sources and context sinks. The role of context management systems is to store, retrieve and evaluate context information and to make it available to applications via synchronous and asynchronous interfaces. Context information is gathered from context sources. In pervasive computing environments, context management systems can be accessed by applications executing on wired and wireless devices as shown in Figure 1. The presence of wireless devices brings a range of challenges, notably an increased frequency of disconnections.

1.2 Scope of the Paper

This paper addresses the problem of disconnections in context aware systems. Disconnections in a context aware system may occur because of node mobility, network failures or node failures.

A research opportunity lies in improving the robustness of context aware systems to disconnections. While traditional distributed systems methods of improving robustness in the face of disconnections can be applied to context aware systems, the additional metadata available to context aware

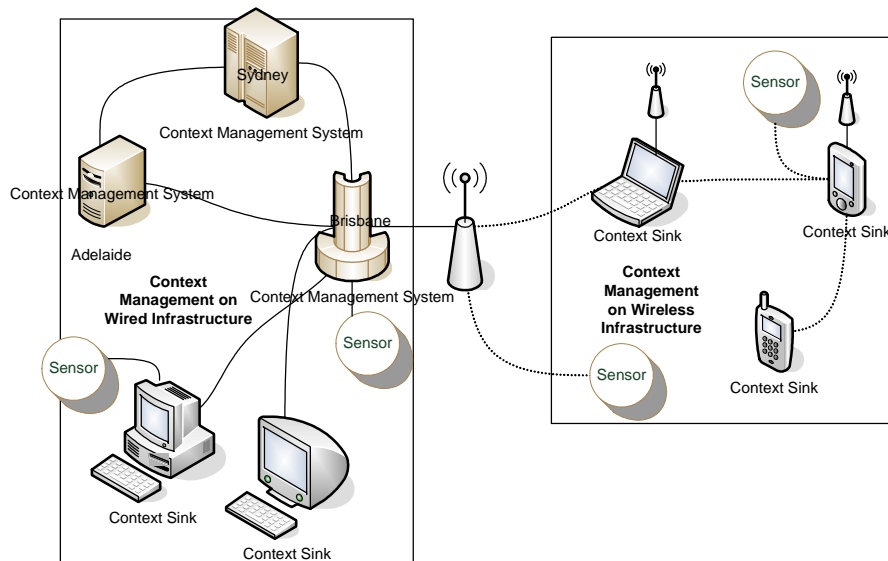


Figure 1: Context Aware System

systems may be leveraged to provide smarter caching algorithms. This research will test the above hypothesis by pursuing research into “smart” caches.

These caches will reside on nodes that contain context aware applications. These nodes will be referred to as clients in this paper. The “smart” caches form part of the context management middleware that runs on the client. The cache’s role is to minimise the impact on context aware applications when the client becomes disconnected from the context management system. To be effective, when disconnected, the cache must maximise the hit rate of requests from context aware applications. The cache must work with a limited amount of space on which it can cache context information.

The remainder of this paper is organised as follows. Section 2 covers background information for the research discussed in this paper. Section 3 covers the design of a “smart” cache. Section 3.4 discusses the test harness which will be used to evaluate the effectiveness of the algorithms that will be used by the smart cache. This test harness will also be used to compare the “smart” algorithms to the current “state of the art” in caching. Section 4 summarises the paper and describes future work.

2. RELATED WORK

2.1 Context Models

Context management systems gather, process and evaluate context information. In recent systems, the gathered information conforms to a particular *context model*. The context models provide some characteristics of context information. Another important role of context models is to bring tried and tested software engineering approaches to the development of context aware applications.

Context models describe the objects and relationships of interest to context aware applications. There is a number of existing approaches for modelling context information. These modelling approaches vary in complexity and functionality. The simplest representation of context that can

be used is the key - value pair [12]. Other approaches are based on ontologies [5, 13, 14]. However, we focus on the Context Modelling Language (CML) [7] as it is the richest and most complete approach to date.

The Context Modelling Language is based on Object Role Modelling (ORM)[1]. An example of CML is shown in Figure 2 (figure reproduced from [6]). The fact type is the basic modelling construct of ORM. Fact types represent the relationships between objects e.g., ‘person located at position’, and are represented graphically. Object types contain a name describing the object. Each fact type in a CML model can be annotated with quality parameters. If a device trying to access context information encounters several copies of the required context information it can choose the information which has the highest context quality.

CML distinguishes four classes of fact types. These fact types are static or dynamic. Dynamic fact types can be further categorised as profiled, sensed or derived. CML provides a method for representing temporal fact types. CML is capable of modelling conflicting information via alternative fact types. Multiple conflicting facts can be stored and resolved when further information becomes available. Fact type dependencies can also be modelled in CML.

While CML is a conceptual device that enables humans to model the relevant objects and relationships in a context aware system, it is not directly suitable for use by a context management system. Instead, a context management system requires a runtime representation of context models and facts. Some groups have investigated the use of ontology-based approaches [5, 13, 14]; however, the determination of appropriate runtime formats for context management systems is still an open research problem.

2.2 Current Caching Techniques

Caching involves temporarily keeping copies of information and using these to carry out subsequent accesses. Caching techniques can be used by nodes of a context aware system to provide applications with context information even if they are disconnected from parts of the network.

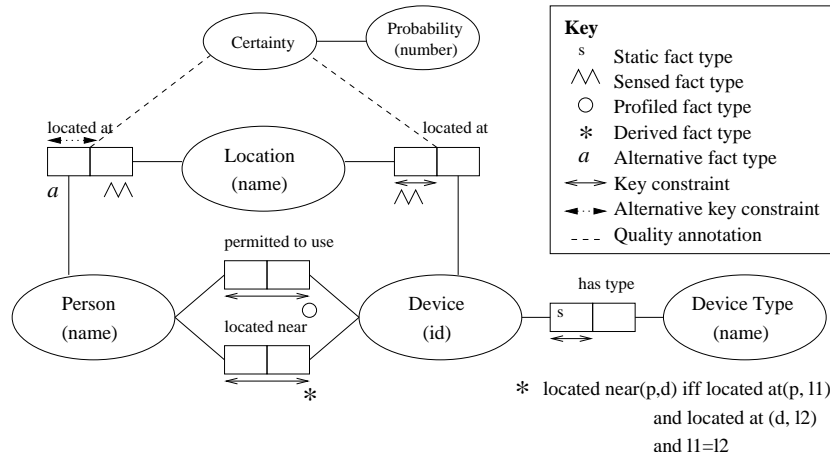


Figure 2: Example Context Model

Client oriented caching solutions [10, 4] are implemented close to the client. There are three reasons for using this type of caching. The first reason is to improve access latency; the second is that they can reduce bandwidth on the network and the third is that if the user becomes disconnected from the server they may still be able to access information in a cache.

Traditionally, information was removed from the cache based on expiry algorithms. These algorithms determine which information to remove based on age, size, and access history. Least Recently Used (LRU) and Least Frequently Used (LFU) are two common expiry algorithms whose functions are reflected in their names.

Modern caching algorithms for mobile devices cache data based on temporality, locality and priority [11]. A temporality based caching algorithm caches information associated with a period of time. This period of time is based on the user's needs. Such an algorithm could be useful in an application used by a fieldworker who is usually out in the field between 11am and 3pm. She has a mobile device which will not be connected to a network while she is away. The algorithm could cache information on the device associated with the time the field worker will be away from her desk.

Location based caching algorithms cache information related to a particular location. For example such an algorithm may only cache information in the vicinity of the user. Others may only cache values related to information in the direction the user is walking in.

Priority based algorithms associate information with a priority. The priority value could be obtained from the user. The priority value can also be derived from the history of access and update patterns.

The caching algorithms discussed in this section are designed for general types of information. This research aims to design a more effective caching algorithm for context information than what can be achieved from using current state of the art in caching. This can be achieved with the aid of CML models and metadata available about context information.

3. DESIGN OF A “SMART” CACHE

As mentioned in Section 1.2 the goal of this research is to minimise the impact on context aware applications when the

client they are running on becomes disconnected from the context management system. The goal of traditional expiry algorithms such as LRU or LFU is to keep information with the highest probability of access [3]. Part of this research aims to prove that a more optimal strategy is to place information in the cache which has the highest ratio between its probability of access (PA) and update frequency (UF). The ratio can be represented as PA/UF. It is argued that update frequency must also be taken into consideration since it is an indicator of the probability that information will become stale. Information which is more likely to become stale should be given a lower priority to be cached.

The success of the caching algorithm will be dependent on two things. The first is how well it predicts which instances in the context management system the context aware application will use in the future (probability of access). The second is how well it predicts which instances are most likely to become stale if the client becomes disconnected from the context management system (frequency of updates). Instances which are more likely to be used should be given a higher priority to be cached.

Upon starting up a context aware application must pass two sets of information to the middleware which can be made use of by the “smart” algorithms to determine probability of access and predict frequency of updates. This information being the CML model it uses and a template. The template contains information which cannot be represented in the CML model. This information represents context aware application preferences which will help determine probability of access. Information stored in the templates may be obtained from the context aware application designer, from the user or a combination of the two. The information that will be contained in the template will be described in more detail in this section and Section 4.

The research undertaken thus far has identified factors which can be utilised by “smart” caching algorithms. A number of candidate “smart” caching algorithms will be designed that differ in the factors they utilise. The algorithms differ in the priority functions they use. One such priority function will be discussed in Section 3.3. These “smart” caching algorithms will be evaluated against each other to test their effectiveness. How the effectiveness of the caching algorithms is evaluated is described in Section 3.4.

3.1 Features of CML that can be Utilised by “Smart” Caches

This section describes features of CML which can be utilised by a caching and purging algorithm to improve caching performance.

Features that can be used to help in the determining access probability are temporal fact types quality metrics and alternative fact types.

The start and end times of instances of temporal fact types can be used to determine if that instance should be cached or purged. A user may only be interested in context information within a specified time period. The temporal information can be taken advantage of to cache only instances associated with this time period.

Quality metrics can also be used by “smart” caching algorithms. A quality metric identified by Henricksen [8] is “freshness”. The “freshness” quality metric describes how current information is. Hence this quality metric can be used by a “smart” purging algorithm to remove expired context data from the cache.

Alternative fact types allow for the possibility of storing multiple conflicting facts. Quality metrics may help the context aware application to determine which of these conflicting facts it should use. Information describing how the context aware application makes this choice should be passed to the middleware so it can determine which of the facts to cache.

Users and/or designers could provide threshold values for each quality metric that appears in a CML model. The caching algorithm will not cache any fact instances with a quality lower than this threshold.

In CML fact types can be classified as static, sensed, profiled or derived. These classifications of fact types can be used to predict update rate. A caching algorithm should treat each class of fact type differently since they are each associated with different update frequencies. Classes of fact types which are likely to be updated frequently are more likely to become stale in the cache. These classes of fact types should have a lower caching priority.

3.2 Parts of the “Smart” Cache

The “smart” caching algorithms operate in one of three states: hoarding, emulation and reintegration. The algorithm is normally in the hoarding state. When a disconnection occurs it will transition into the emulation state. In this state requests from context aware applications are satisfied by the cache. Since server misses cannot be serviced or masked, they appear as failures to context aware applications and users. When reconnected it will transition into the reintegration state. In this state the middleware running on the client resynchronizes its cache with the context repository.

There are four main components of the “smart” caching algorithm. These components being: Hoarding algorithm; purging algorithm; recording algorithm and resynchronization algorithm. The hoarding, purging and resynchronization algorithms make use of the priority function. Figure 3 shows a logical view of the system.

The recording algorithm keeps track of the number of times instances of the different fact types are accessed. The recording algorithm also keeps track of the number of times an instance is updated. The hoarding algorithm determines if an instance should be added to the cache. The purging

algorithm removes instances from the cache which have become stale. The resynchronization algorithm runs in the reintegration state. This algorithm coordinates with the context management system and updates the cache appropriately with instances updated during the disconnected period.

The measurements from the recording algorithm will be used to calculate access rate and update rate. These rates should represent the recent number of accesses and updates. The period of time used to measure these values will be specified by the user. If these rates influence the priority function the priority values of the instances will change. In an example case where the period of time is one hour, priority values will be based on recordings from the previous hour. The cache and the context management system are said to be unsynchronised if there is an instance in the context management system which satisfies all the following three requirements: it has not been cached; has a higher priority than at least one instance in the cache; and satisfies the quality requirements. A change in priority values may cause the middleware to become unsynchronised. The cache and the context management system will be resynchronised at the end of the recording period. By synchronising at the end of the period and not continuously, communication and processing overhead will be reduced. The resynchronization is carried out by the resynchronization algorithm.

The hoarding algorithm runs in the hoarding state each time the cache is notified of a change in the state of the context management system. The hoarding algorithm must resynchronise the cache and context management system if required. If an instance in the context management system has a higher priority than what is already in the cache and the instance satisfies the quality requirements it will be placed in the cache. If there is no room to place such an instance the hoarding algorithm purges the instance in the cache with the lowest priority value. An instance can also replace one which conflicts with it in the cache if it satisfies the quality metrics more.

The purging algorithm is called at regular intervals and runs during the hoarding state. The purging algorithm purges instances which have become stale. The purging algorithm replaces the same number of purged instances with new instances. These new instances are the ones with the highest priority in the context management system which are not yet in the cache.

3.3 Candidate Priority Function

A candidate priority function that will be used by the “smart” caching and purging algorithms will be discussed in this section. It only utilises the factors PA and UF. Other candidate priority functions designed later will make use of other factors. The equation shown in this section is based on the hypothesis that a priority function based on the PA/UF ratio will yield a better result than one based on probability of access alone. The priority function also utilises the fact that UF is inversely proportional to probability of freshness (PF). The priority function is shown in Equation 1.

The priority value of each instance is tracked on the context management system. The instances with the highest priority values will be cached. The cache will always hold as many instances as possible.

$$\text{Priority Value of Instance} = \text{PA} \times \text{PF} \quad (1)$$

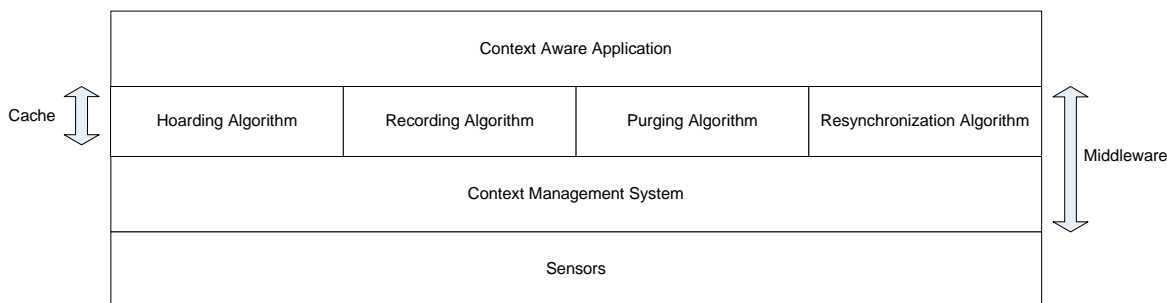


Figure 3: Logical View of System

| | |
|-----------|----------------|
| Static: | PF = 1 |
| Profiled: | PF = 0.75 |
| Sensed: | PF = 0.5 |
| Derived: | PF = will vary |

Table 1: PF Values

For this candidate priority function PA is a ratio of the number of times instances belonging to the same fact type, as the fact type being evaluated, are accessed compared to all accesses.

PF will be assigned values based on fact type classification for this candidate priority function. Other candidate priority functions which use PF may have a different strategy to assign a value to it. The probability value associated with each fact type classifications is provided by the context aware application designer. PF represents the probability that if the instance is in the cache and the client has been disconnected from the context management system that it will be fresh. Table 1 shows values assigned to PF for the different fact type classifications. The value of PF for derived fact types will depend on fact type classifications of the information it is derived from.

3.4 Test Harness

The test harness will be implemented later in the research around the “smart” algorithms to evaluate their effectiveness. A high level diagram of the test harness is shown in Figure 4. This test harness exercises the caching algorithm by emulating the context aware applications. Disconnections will also be simulated by the test harness. The algorithms’ effectiveness will be measured in terms of hit rate, consistency of context information and bandwidth utilisation. The cache hit rate is a percentage of the results of a query that have been successfully found in the cache compared to the total queries made. Measuring consistency involves analysing what was retrieved from the cache with what is actually in the context management system. The bandwidth utilisation will be measured in two ways, these being the number of queries made to the context management system by the caching algorithms and the number of bytes transacted between the algorithms and the context management system. The algorithms have a limited amount of space in which they can cache context information. The “smart” caching algorithms will be evaluated with varying amounts of space available to them to cache information. This is done to evaluate if the algorithms’ effectiveness varies for different cache sizes. The algorithms will also be evalu-

ated under different periods of disconnection to see if these different conditions influence their effectiveness.

The same techniques used to measure the effectiveness of the “smart” algorithms will be used on current state-of-the-art in caching. This is done to compare the algorithms to each other. The current state-of-the-art in caching does not take advantage of any of the features of CML. For this research to be deemed successful it must produce “smart” caching and purging algorithms more effective than the current state-of-the-art in caching.

4. FUTURE WORK

A CML model describes the kinds of information that are relevant to an application. However, a particular instance of an application may be interested only in specific fact instances. For example, Bob’s application may be interested only in information pertaining to Bob. Therefore, a proactive caching mechanism must be able to cache only fact instances that are associated with Bob. The application’s interests may be more specific. It may only be interested in information about Bob when he is at work. The role values of interest to the context aware application can be passed to the middleware through the template.

As mentioned in Section 2.2, current caching techniques can be based on location. The CML model can be used to take advantage of this. The “smart” algorithm can be made to cache instances associated with a particular location. This feature can be added by modifying the priority function to give higher priority to such instances. Since the CML model shows us the relationships of different objects we can take advantage of this. For example in Figure 2 if the device is in the area of interest the device type may also be cached. Whether or not device type should be cached will be communicated to the cache using the template.

Guidelines will be defined for context management system designers to follow when modelling CML. These guidelines will help designers maximise the effectiveness of their CML models in their support of the “smart” caching and purging algorithms. This involves defining a process that context aware system designers can follow to identify which quality metrics should be included in the CML models they design. NICTA¹ is developing a context management system named the NICTA Context Engine (NICE). The caching algorithm will be integrated into the NICE. The modified NICE will be evaluated in a real life scenario.

¹<http://www.nicta.com.au>

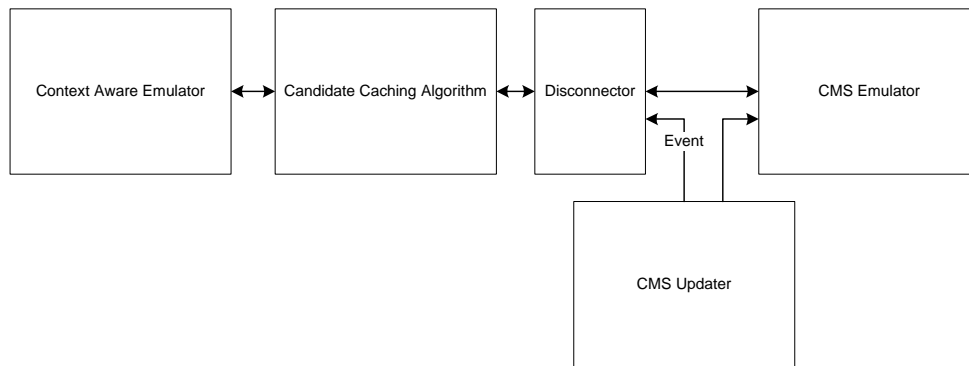


Figure 4: Test Harness

5. SUMMARY

Context aware systems are made up of context aware applications, context sources and context management systems. The context management system is middleware between the context aware applications and context sources. While traditional distributed systems methods of improving robustness in the face of disconnections can be applied to context aware systems, the additional metadata available to context aware systems may be leveraged to provide smarter caching algorithms. “Smart” caching algorithms utilise metadata provided by CML models. Features of CML that can be used by the “smart” caching algorithm are representation of static and dynamic data, temporal fact types, quality metrics, and alternative fact types. It was also hypothesised that the ratio between an instance’s access probability and update frequency will yield a more optimal priority value than one which considered access probability alone.

6. ACKNOWLEDGEMENTS

National ICT Australia is funded by the Australian Government’s Department of Communications, Information Technology, and the Arts; the Australian Research Council through Backing Australia’s Ability and the ICT Research Centre of Excellence programs; and the Queensland Government.

7. REFERENCES

- [1] Object role modeling, <http://www.orm.net/>, April 2006.
- [2] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggle. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer-Verlag, Karlsruhe, Germany, 1999.
- [3] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: data management for asymmetric communication environments. pages 199–210, 1995.
- [4] A. Chankhunthod, P. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical internet object cache. In *USENIX Technical Conference*, pages 153–164, San Diego, CA, USA, 1996.
- [5] H. Chen, T. Finin, and A. Joshi. *The SOUPA Ontology for Pervasive Computing*. Ontologies for Agents: Theory and Experiences. Springer, 2005.
- [6] K. Henriksen and J. Indulska. Modelling and using imperfect context information. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference*, pages 33–37, 2004.
- [7] K. Henriksen and J. Indulska. A software engineering framework for context-aware pervasive computing. In *Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference*, pages 77–86, 2004.
- [8] K. Henriksen, J. Indulska, T. McFadden, and J. Purser. Developing context aware applications for smart phones. Technical report, 2005.
- [9] M. Korkea-aho. Context-aware applications survey, Department of Computer Science, Helsinki University of Technology, <http://users.tkk.fi/mkorkea/doc/context-aware.html>, 2000.
- [10] I. T. Kuz. *An Approach to A Scalable Wide-Area Web Service*. PhD thesis, Technische Universiteit, 2003.
- [11] I. Mahgoub and M. Ilyas. *Mobile Computing Handbook*. CRC Press, Dec 2004.
- [12] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. *IEEE Workshop on Mobile Computing Systems and Applications*, 1994.
- [13] T. Strang, C. Linnhoff-Popien, and K. Frank. CoOL: A context ontology language to enable contextual interoperability. In J.-B. Stefani, I. Dameure, and D. Hagimont, editors, *LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003)*, volume 2893 of *Lecture Notes in Computer Science (LNCS)*, pages 236–247. Springer Verlag, Paris/France, 2003.
- [14] Z. Wang, D. Zhang, T. Gu, J. Dong, and H. Pung. Ontology-based context modeling and reasoning using OWL. *Context Modeling and Reasoning Workshop at PerCom 2004*, pages 18–22, 2004.