An Approach of Semantic Cache for Mobile Devices to Enhance the Performance of Applications

Marcos F. Caetano[1], Marco Antonio Ribeiro Dantas[2]

[1] Computer Science Department (CIC)
[1] University of Brasilia (UnB), 70910-900, Brasilia, Brazil
[2] Department of Informatics and Statistic (INE)
[2] Federal University of Santa Catarina (UFSC), 88040-900, Florianopolis, Brazil
[1] caetano@cic.unb.br
[2] mario@inf.ufsc.br

Abstract: Wireless mobile computing has become a differential aspect to a large number of distributed applications. The main research goal related to this subject is to provide to applications a similar level of services found in structured networks. An example of interesting research topic is to improve the data replication scheme that exists in a mobile device. The use of an elaborate method can represent an improved strategy to enhance the performance and availability of applications to wireless users. In this article we present a semantic cache model study, and its implementation, as a differentiate policy for the local management of data replication. The approach assumes that an answer for a query can be satisfied totally by a local semantic segment, considering the gathering action of partial segments or helping the connection to a server and then receiving the answer. We tested the implementation of the model in a real environment, assuming three classes of queries. The first class was characterized by those queries that could be promptly answered locally. The second type of queries was answered after a local gathering operation of semantic segments. In the last situation, queries were answered after a send-receive operation to a server node in a structured network. The empirical results indicate that the approach has improved successfully the performance of the applications, avoiding unnecessary connections to a server when an answer could be reached using one local cache segment, or gathering information from the local semantic segments to compound the answer. In the case of inexistence of any local semantic segment information to reply a query, the implementation works transparently to connect to a server and then answer the wireless user.

1 Introduction

The fast growing rate of new mobile computing technologies has brought many benefits for wireless users. On the other hand, this fact promotes several research challenges to provide software services in a similar level found in structured networks.

Investigation efforts such as [CM05, HMM05, DM04] target to prove a performance enhancement of applications based upon the location of devices, efficiency of communication strategies between client and server, or the treatment of replicas and its reconciliation.

In contrast to the former researches initiatives, our research work has the goal to enhance the performance of mobile applications using a model of cache policy. The implementation of this paradigm was based on semantic cache. The main objective of this approach is to prove the right answer to queries using as much as possible the information stored at the local device. In other words, we first search for a complete or partial solution inside the mobile device. Therefore, the solution avoids unnecessary wireless communications with a server node. In the case that an answer could not be found correctly a communication is established to a specific node that will provide the answer for the query.

This paper presents the model that we have adopted and its implementation, considering real mobile devices in some case study configurations. Experimental results of the approach indicate that the proposed paradigm implementation has successfully reached the objective for a number of cases studies.

The article is organized as follows. In section 2 we present the fundamental theory related to semantic caches in mobile devices. In addition, in this section we also show some related work. The experimental environment employed for our case studies is illustrated in section 3. Finally, in section 4 we draw some conclusions related to the present research and point out some future work.

2 Semantic Cache

Semantic cache can be broadly defined as a collection of semantic segments [Sd96]. Each segment has a result of a query and its description, which was previously executed [AJ96]. Utilising the existing description it is possible to execute operations and verify whether a new query can be answered by this segment, or not.

Semantic cache has been used in centralised systems [CN94, Nr91], client-server environment[Sd96, AJ96], OLAP systems [Pd98], heterogeneous systems [PJ97] and also in mobile computing environments [KHA99, QM99].

When a query Q is submitted to a semantic cache approach, the entity responsible for the management of the cache verifies if a semantic segment of the entity can answer the query. This procedure is repeated for all queries, the semantic mechanism indicates if a result can be reached or not. In the case of a partial answer exists, the initial query is divided in two parts. The answered portion is associated to a segment and later is processed. On the other hand, the part that was not answered is submitted again for the next semantic segment.

The procedure is repeated until an answer is found to the second part inside the mobile device, or there is no more semantic segments to process the query. In this case, the second part of the divided query is submitted to a server node to be processed. It is expected that in the end, all parts are gathered and initial query Q answered.

It is interesting to mention that the new data that came from the server node will form a new semantic segment that can be used locally for future answers.

2.1 Cache Structure

Each semantic segment that forms a cache, it is defined by a tuple from five subsets[Qr00]:

$S = <S_R, S_A, S_P, S_C>$

Where, the above parameters, related to the *SQL* language, have the meaning:

- S_R – source relation that translates how the segment is form, thus it contains names of the tables declared in the section *from <table>*;
- S_A – it represents a set of attributes that are specified in the *select <atrib1,atrib2,....>* section;
- S_P – a set of predicates that are specific for the *where <pred1,pred2,...>* section;
- S_C – it represents a set of answers from segment *S*.

S	S_R	S_A	S_P	S_C	S_{ts}
S_1	Employee	{Id,Name, Age}	Age > 30	2	T_1
S_2	Employee	{Id,Name, Phone}	Age < 20	5	T_2

Table 1: An example of a semantic cache structure

Table 1 illustrates a structure model example that implements a cache segment. Segments S_1 and S_2 where created from the following queries:

- Q_1 *select* Id, Name, Age *from* Employee *where Age* > 30;
- Q_2 *select* Id, Name, Phone *from* Employee *where Age* < 20;

S_C represents a pointer that indicates the number of the page that contains the result of the semantic segment. On the other hand, S_{ts} is the representation of the *time stamp* that keeps the information time when the segment was created.

2.1 Query Partition

When a query is submitted to the local cache, a number of procedures will be realised targeting to solve this query. All the segments that form the cache can have a portion that together can answer the question. Therefore, a semantic segment *S* can contribute entirely, partially or not contribute at all to solve one question *Q*. Important to observe that even for the case that a small fraction of the question can be answered using a semantic segment this fact represents less wireless communication. In addition, if we avoid unnecessary communication we are also helping to provide to the mobile device a battery save feature.

However, in the case of partial answer the procedure will necessarily execute the division of the *Q* question into two queries:

- *Probe* – part of the query *Q* that can be answer by the semantic segment;
- *Remainder* – the present semantic segment could not answer this portion of the query *Q*.

Figure 1: Examples of queries submitted to a semantic segment *S*.

Figure 2 has a graphical illustration of five possible cases that a query can be classified. Each case can be understood using the examples shown in figure 1. Query Q_1 submitted to the *S* segment represents *case 1*, Q_2 translates the *case 2* and so forth. In [Qr00, QM03] it is possible to find a more formal description of these concepts.

The five cases presented in figure 1 can be understood as:

- *Totally Contained:* this represents the first case, when a query Q_1 is submitted to segment *S*. It is possible to verify $Q_P \Rightarrow S_P$ and $Q_A \subseteq S_A$. In other words, query Q_1 selects the same attributes of *S* (*Age,Name*) and the predicate Age > 38 is a subset that entirely exists inside the predicate Age > 24 from segment *S*. As a result, the answer for the query Q_1 is completely inserted in *S* segment;

- *Horizontally Partitioned*: When a query Q_2 is submitted to the segment S, it is possible to verify that $Q_A \subseteq S_A$ and $Q_P \wedge S_P$ satisfies the question. This fact can be understood because Q_2 selects the same attributes set from S, and the predicate *Age > 17* finds inside the segment results where values are greater than 24. In this case, the predicates from the query Q_2 are divided into two parts:
  - $Q'_2 = Q_p \wedge S_p$: a Q_2 subset that requires existing values inside S, i.e. values greater than 24;
  - $Q''_2 = Q_P \wedge \neg S_P$: a Q_2 subset that requires values that do not exist inside S,i.e. value between 18 and 24.

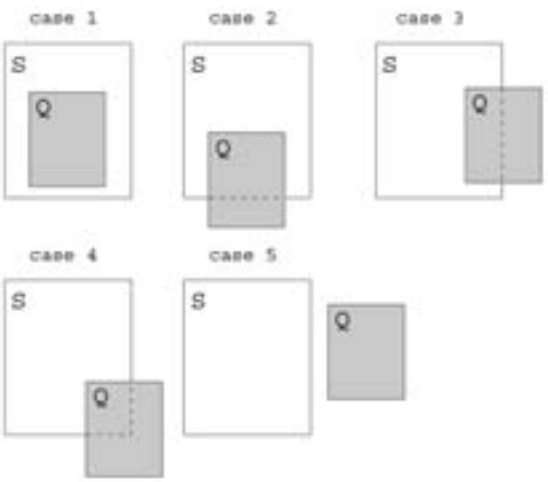

Figure 2: Possible partition semantic segment for query Q.

The Q'_2 query, called *probe query*, it will be entirely processed inside the segment *S.* In contrast, Q''_2, the query known as *remainder query*, will be redirected to other segment. In the case that no existing segment from the local cache could answer this query, it will sent to a server to be processed. Figure 3, presents the entire algorithm definition of the *query trimming*. In this section, we simplified the notation targeting to help the understanding of the idea.

- *Vertically Partitioned*: When query Q_3 is submitted to the segment S it is possible to verify that $Q_P \Rightarrow S_P$ e $Q_A \not\subseteq S_A$. Thus, the predicates from the query Q_3 are completely answered by the predicates from segment *S.* On the other hand, the attributes selected from Q_A (*Age, Phone*) do not entirely exist inside the segment attributes S (*Age, Name*). Therefore, the attributes

of the query Q_3 will be divided into two parts:

- $A_1 = (Q_A \cap S_A) \cup K_A$: compound by a subset of attributes that exists inside segment S (*Age*);

- $A_2 = (Q_A \cap \neg S_A) \cup K_A$; this part translates the attributes subset that does not exist inside the segment S (*Phone*).

The two subsets A_1 and A_2 will respectively form the *probe* and *remainder queries.* As we discussed in previous cases, the *probe query* will be executed locally. In contrast, the *remainder query* will be submitted to the other semantic segments. In the case that no local segment could match the query, it will be necessary a communication to a server to provide a answer. In the end, the subset will be gathered to solve the query Q_3. The concatenation of the subsets is done using the primary key from table *Employee.* Every query has the primary key of the manipulated table as an attribute. This attribute is defined as *included key segment* [QM03].

- *Hybrid Partitioned:* This approach can be considered the most complex, when it is compared to the previous cases. The reason for the complexity is the necessary utilisation of both *horizontal* and *vertical* partitions, respectively. When a query Q_4 is submitted to the segment S, it is possible to observe that:
  - $Q_p \wedge S_p$ can find a suitable answer. The predicate *Age > 17* finds inside the segment only values greater than 24. Therefore, values from the threshold 18 and 24 are not found*;*

  - $Q_A \not\subset S_A$ represents attributes selected from Q_A *(Age, Phone)* that are not completely inserted inside the segment *S (Age, Name);*

  - *Probe query*: a subset that represents values that exists in the segment S, i.e. $PQ = \pi\ (Q_A \cap S_A) \cap K_A \sigma\ Q_P \wedge S_P(Q_R)$. In other words, the attribute *Age* and the values of *Age > 24* are obtained in the segment.

- *Not Contained*: The query Q_5 can not be answered by segment S, i.e. $Q_P \wedge S_P$ is not satisfactory. Thus, the *probe query* is empty and the *remainder query* represents the query Q_5. This query will be processed by the next semantic segment until it can be solved. In the case that no answer is possible to be found from any segment, then the query is submitted to a server node.

The previous five described cases are organised in an algorithm structure presented in figure 3. The algorithm *Query Trimming* [QM03] receives as an entrance a semantic segment *S* and a query *Q*. As a result, the algorithm provides a structure that contains: a *type identifier* (that specifies which kind of partition was selected) and *sub-queries* (probe, remainder and amending query). An *amending query* is characterised by a query sent to a server node. In other words, this query represents the operation to find a segment that it is not present in memory of the mobile device.

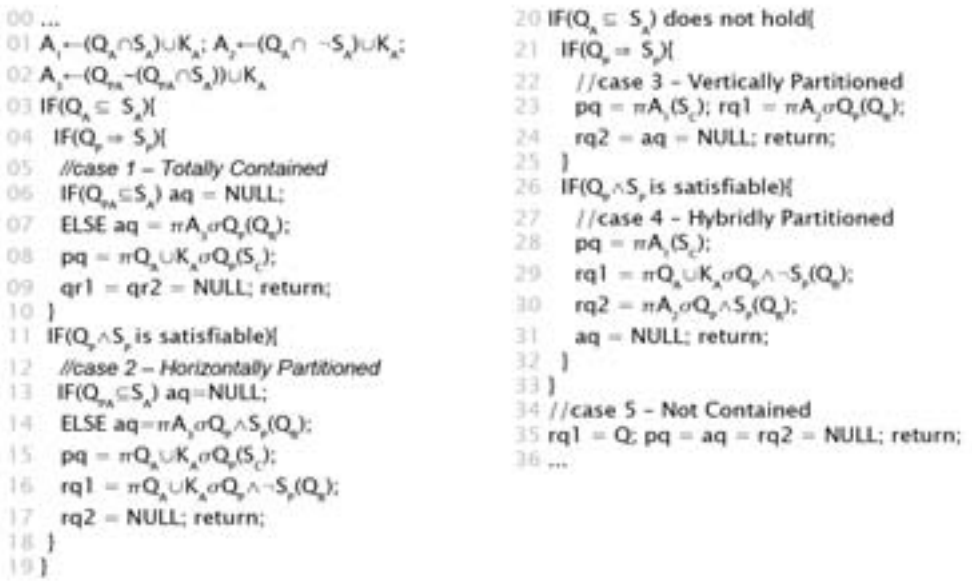

Figure 3: Algorithm Query Trimming.

In the following query structure we provide an example, where query *Q*' submits a search to segment *S*. A answer can not be found only using the local information available in the cache:

```
select Age, Name from Employee where Age > 30 and Salary > 300
```

It is possible to verify that the condition related to the employees, i.e. with Age > 30 and Salary > 300, can both be solve by the segment *S*, if the attributes exist in the local segment. However, if an attribute does not exist it will be impossible to solve the query. This example fits exactly in the case, because the local segment does not have the *Salary* attribute. Then, an *amending query* is created. This query submits to the server node a query to send to the mobile device the *Salary* attribute. When this attributes arrives at the local segment the query is processed and the result returned to the wireless user.

3 Configuration and Experimental Results

The environment considered in this article, it is a real configuration form by mobile devices that can have access to a structure local area network. We decided to utilise a real configuration, because we implemented the software portion of the server, the client and the communication module. Therefore, utilising this configuration it could provide a more realist assessment of some intrinsic problems found in wireless environments.

The design and development of the semantic cache model, presented in the theory in section 2, represents the main part of our contribution. As figure 4 shows, we image an environment that was formed by three elements: a database manager, a server and a client wireless node.
The database manager used in this research project was the Mckoi SQL [Md06]. We decided to use this database, because it is an open source package and provides all the functions necessary to the development of the proposed software environment.

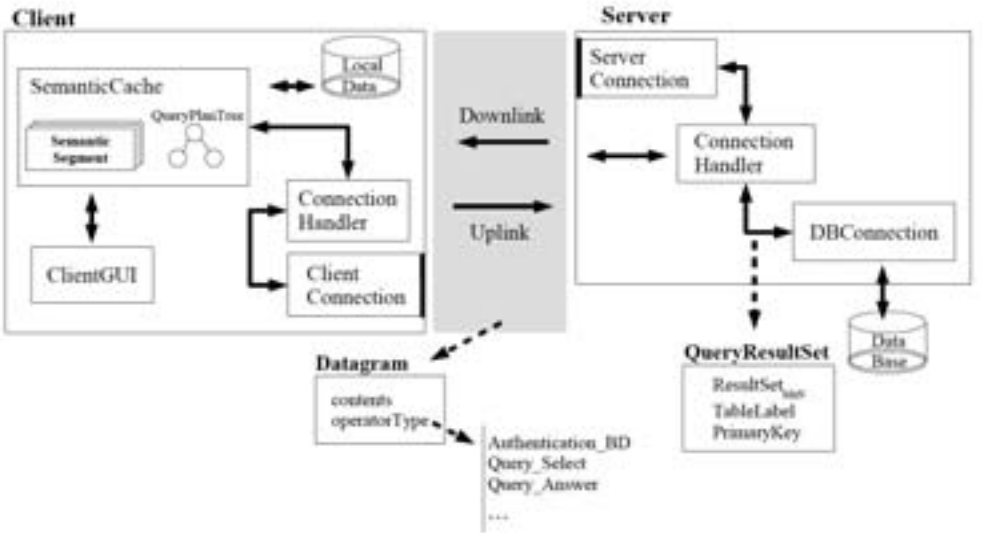

Figure 4: Proposed Environment.

The server node module was characterised for waiting and treating client connections and requests. This element was implemented having three software components:

- *Server Connection*: this element waits for the calls from wireless clients in a specific port and then redirects to the *Connection Handler* module to be treated;
- *Connection Handler*: this module is responsible for receiving calls from client nodes and establishes connections with the DBConnection. It was implemented adopting the multithread approach and has an application communication protocol developed to provide some functions such as:

authentication in the database, execution of queries and database re-initialisation.

- *DBConnection*: this element interfaces the communication between the server node and the database manager. The implementation was developed to provide to the server independence from the used database. Therefore, this module returns the results of all SQL queries as *labels*, indicating a line and a column and the primary key of the table. In other words, this software element was designed adopting a *QueryResultSet* structure.

On the client side, the enhanced semantic cache functionality was designed and implemented having the following components:

- *Client Connection***:** this element is responsible for establishing and closing a connection, sending and receiving requests in the client connections. These connections were characterised by *sockets* and this module provides two streams channels: an input and an output. Theses channels are managed and encapsulated through the functions:

```
public void send(Datagram msg) {...}
public Datagram receive() {...}
```

- *Connection Handler***:** this entity is responsible for treating the client connections. Thus, it provides a transparent interface, called as *Semantic Cache*, that encapsulates procedures to establish connection with a server and data structure;
- *Semantic Cache*: this module is the implementation of semantic cache policies that we discussed early. Therefore, we developed for this component some facilities such as: semantic segment, binary tree, QueryPlanTree, and PredicateSet. In addition, it also provides the entity *ClientGUI* that has a transparency feature related to the semantic cache policy. Thus, for an application that uses this interface, the processing procedure for local and remote queries are transparently. An example is:

```
public synchronized QueryResultSet
        executeQuerySelect( String queryConsult)
        {…}
```

- *ClientGUI:* this component of the proposed environment provides a user friendly interface to a user to have access to the facilities of the software package implemented;
- *Local Data:* the persistent local cache is store in this element. When, for example, the environment starts it has a number of segments that were generated in the last utilization.

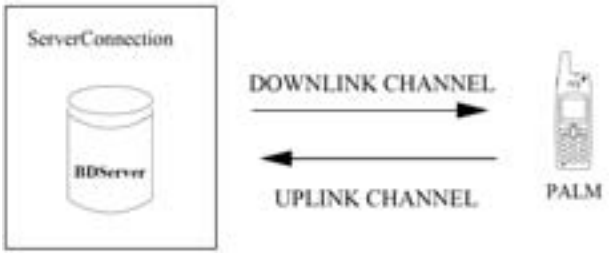

Figure 5: A general communication between client and server.

Figure 5 presents a general picture of the client and server communication. The *uplink* channel is used by clients to realize access and queries to a server node. This channel is also used to verify if enough energy exists and if the client is inside a wireless area from a server. The *downlink* channel is a conventional link where servers send their answers to clients.

Table 2 shows characteristics of the software and hardware components that were used to our empirical tests.

Component	Mobile Client	Server
Model	Palm Tungsten C	AMD Duron
Processador	400MHz	1,2GHz
Memory	64MB	256MB
OS	Palm OS 5.2.1	GNU-Linux-2.4
JVM	MIDP 2.0 - CLDC	Sun J2SE-1.4.2

Table 2: Environment Characteristics

3.1 Empirical Experiments

The experiments that we show in this section were realized to test all the existing components of the environment, shown in figure 4. In other words, we image a set of tests that could verify the interaction between the software elements and certify that the answers were expected results.

We started the mobile device with any information inside the local cache. After that we submitted the following queries:

1. Select Id, Name, Age from Employee where Age > 35;
2. Select Id, Name, Phone from Employee where Age < 40.

This first query was entirely submitted to the server and after that the information was added to a segment S_1. The second query, shown in figure 6, when submitted to the first segment S_1 generated a *hybrid partitioned*. The sub-queries were:

1. Select Id, Name from Employee where Age > 35;

2. Select Id, Phone from Employee where Age > 35 and Age < 40;
3. Select Id, Name, Phone from Employee where Age <= 35.

The first sub-query can be answer by the segment S_1, whereas the other two were sent to the server node. An example of the server behavior is illustrated in figure 7. This figure shows a server text interface informing the result sent the client. This message refers to the third sub-query.

The final answer of the second query Q is present in figure 8. This answer indicates that the designed and implementation reach successfully the objective of the wireless user. The research procedure was efficient, because it only collects the necessary data from the server node. In other words, avoiding unnecessary wireless connection to the server and saving resources once it does not considered the store of data that is already in a semantic cache segment.

On the other hand, figure 9 demonstrates the final content of the local semantic cache. After the *hybrid partitioned* operation two new semantic segments were created, S_1 *and* S_2. It is expected that for a future similar request, the mobile device could answer the query more efficiently, because the data now exists locally.

4 Conclusions and Future Work

The semantic cache is not a novel mechanism. However, this approach employed in small equipment, such as mobile devices, is an interesting research topic to enhance wireless applications. In this article we have presented a model and implementation of an environment to enhance the performance of mobile applications. We first described theoretically how the semantic segments work. After that we presented our differential contribution designing, developing and implementing a solution to receive queries from a local mobile device and then process efficiently these requests.

The proposed environment was compound by three software modules: a server, a client and a database interface manager. The server was implementing having entities such as a connection link element, a handler and a database connection. The client part representing the vital point of the contribution, it was characterised by having a connection server software package, a user friendly graphical interface and the semantic cache module. Finally, the database interface manager provides an open solution to the environment to work independently from a specific database or operating system. Empirical results indicate that we have successfully reached a contribution to improve the performance of local wireless applications.

As a future research work, we are planning to add locality dependence facility to the software package implemented and also consider *ad hoc* networks to form an environment to provide answers for mobile users.

Figure 6: The second query from the mobile device.

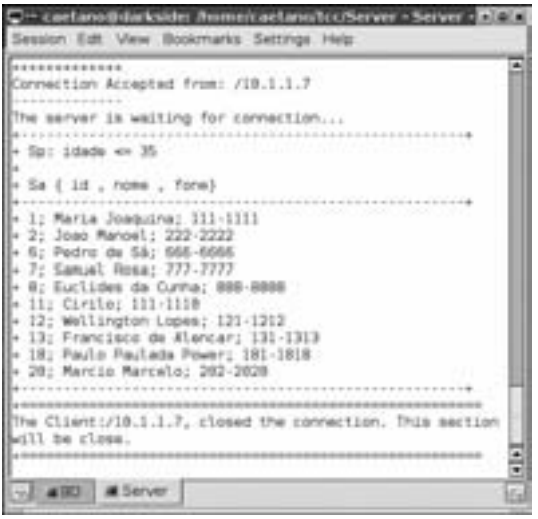

Figure 7: Server Text Interface.

Figure 8: Answer of the second query.

Figure 9: Semantic segment after the second query.

References

[AJ96] A. M. Keller and J. Basu. A predicate-based caching scheme for client-server database architectures. Parallel and Distributed Information Systems, 1994., Proceedings of the Third International Conference on, 5(2):35–47, September 1996.

[CM05] C.D.M. Berkenbrock and M. A.R. Dantas. Investigation of cache coherence strategies in a mobile Client/Server environment. *International Conference on Computational Science*, (3):987–990, 2005.

[CN94] C. M. Chen and N. Roussopoulos. The implementation and performance evaluation of the adms query optimizer: Integrating query result caching and matching. *In Proceedings of EDBT*, pages 323–336, march 1994.

[DM04] D. Pezzi and M.A R. Dantas. An experimental case study of replication and reconciliation in a wireless environment. Proceedings of The 18th International Symposium on High Performance, pages 179–182, 2004.

[HMM05] H. Monica, M. S. de Camargo, and M. A. R. Dantas. An architecture for location-dependent semantic cache management. ICEIS, (1):320–325, 2005.

[KHA99] K. C. K. Lee, H. V. Leong, and A. Si. Semantic query caching in a mobile environment. Mobile Computing and Communication Review, 3(2):28–36, 1999.

[Md06] Mckoi Database, http://www.mckoi.com/database/, available in April, 2006.

[Nr91] N. Roussopoulos. An incremental access method for viewcache: Concept, algorithms and cost analysis. ACM Transactions on Database Systems, 16(3):535–563, September 1991.

[Pd98] P. Deshpande, K. Ramasamy, A. Shukla, and J. F. Naughton. Caching multidimensional queries using chunks. *In Proceedings of ACM SIGMOD*, pages 259–270, june 1998.

[PJ97] P. Godfrey and J. Gryz. Semantic caching in heterogeneous databases. In Proceedings of DEXA Workshop, pages 414–419, August 1997.

[QM99] Q. Ren and M. Dunham. Using clustering for effective management of a semantic cache in mobile computing. In Proceedings of the International Workshop of MobiDE, pages 94–101, August 1999.

[QM03] Q. Ren, M. Dunham, and V. Kumar. Semantic caching and query processing. Knowledge and Data Engineering, IEEE Transactions on, 15(1):192–210, Jan.-Feb. 2003.

[Qr00] Q. Ren. Semantic caching in mobile computing. PhD thesis, Southern Methodist University, February 2000.

[Sd96] S. Dar, M. J. Franklin, B. T. J´onsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. *Proceeding of the* 22*TH International Conference on Very Large Data Bases, Mumbai (Bombay), India*, pages 330–341, 1996.

Chapter 2: Decentralized Network Systems

Contributions to 10th I2CS 2010, Bangkok, Thailand

Christian Spielvogel, Peter Kropf
Application Layer Scalable Video Coding for the iPhone

Lada-On Lertsuwanakul
Fuzzy Logic Based Routing in Grid Overlay Network

Miguel Angel Rojas González
Performance Evaluation of two Self-Adaptive Routing Algorithms in Mesh Networks

Oleksandr Kuzomin, Illya Klymov
Functional Approach to Decentralized Search Engine for P2P-Network Communities