# AN ARCHITECTURE FOR LOCATION-DEPENDENT SEMANTIC CACHE MANAGEMENT

Heloise Manica , MAR Dantas
*Department of Informatics and Statistics - Federal University of Santa Catarina (UFSC)*
*Campus Universitario Trindade, Caixa Postal 476*
*Florianópolis – SC, Brasil, CEP 88.040-900*

Murilo S. de Camargo
*Department of Computer Science - University of Brasília (UnB)*
*Campus Universitário Darcy Ribeiro*
*Brasília – DF, Brasil, CEP 70.910-900*

Keywords: Mobile Computing, Mobile Databases, Cache Management, Semantic Cache, Location-Dependent Systems.

Abstract: Location-Dependent Information Services is an emerging class of application that allows new types of queries such as location-dependent queries and continuous queries. In these systems, data caching plays an important role in data management due to its ability to improve system performance and availability in case of disconnection. In mobile environment, cache management requires more than traditional solutions. This paper presents a new semantic cache scheme for location dependent systems based on spatial property. The proposed architecture is called as Location Dependent Semantic Cache Management – LDSCM. In addition, we examine location-dependent query processing issues and segment reorganization.

## 1 INTRODUCTION

Advances in mobile computing make possible the development of new classes of services such as Location-Dependent Information Services (LDIS). These services emerged with a variety of promising applications, such as local traffic report, hotel or restaurant information and emergency services.

The access data, called Location Dependent Data - LDD, are related to their geographical position and the queries that issue for them are named Location-Dependent Queries - LDQ (Ren and Dunham, 2000). A location-dependent query usually originates from a moving client, and the location determines the query results. “Find all restaurants within 10 miles of my position” is an example of a LDQ. The client’s geographical location is provided by a location service or global positioning system.

Data management in an LDIS faces several challenges that have opened many new research problems (Lee et al., 2002). Mobile computing environments offer low-quality communication, frequent network disconnections and limited local resources. In addition, because of mobile user’s movement, some tasks such as data cache management are tough.

Data caching plays a key role in data management in mobile computing since it helps to save power consumed with server communication. Besides, data caching improve data availability in case of disconnection. The mobile users are still able to work using the cached data when the network is unreachable.

In LDIS, cached data become invalid not only as a result of updates performed on data items in the server but also when a mobile user changes its location (Zheng et al., 2002). Traditional solutions are not suitable for LDIS and additional spatial properties such as data distance and movement must be considered. In conventional caching schemes such as tuple-based or page-based it is difficult to explore spatial properties because the cached data are not associated with any semantic meanings.

A better alternative for mobile computing environment is the semantic caching (SC) model (Dar et al, 1996; Lee et al., 1999; Ren and Dunham, 2000; Ren et al., 2003). The semantic cache stores in sets, named semantic segments, results of queries



Manica H., Dantas M. and S. de Camargo M. (2005).
AN ARCHITECTURE FOR LOCATION-DEPENDENT SEMANTIC CACHE MANAGEMENT.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 320-325
DOI: 10.5220/0002524903200325

submitted to the server (Ren and Dunham, 2000). Moreover, the SC maintains semantic descriptions associated to the answers of queries organized in an index structure. The semantic segments are disjoint and each tuple in the cache is associated with exactly one semantic segment. This model is ideal for location-dependent applications since it makes cache management more flexible. With semantic information, strategies with higher cache hit ratio can be developed.

In a previous work (Manica et al., 2004-b), we propose a new semantic cache scheme for LDIS based on spatial property. In this paper, we improved our ideas, propose an architecture named Location Dependent Semantic Cache Management (LDSCM) and we better define our location-dependent query processing issues and semantic segments reorganization.

The remaining of this paper is organized as follows. Section 2 present related previous work. In section 3 we propose the semantic cache model and the algorithms for query processing and cache management issues. The design of our semantic caching architecture is presented in section 4. Finally, the section 5 concludes this article and present future works.

## 2 RELATED WORK

Querying location dependent information in mobile environment has been an important research area. In (Lee et al., 2002) the authors discuss location dependent information access in a mobile-pervasive environment and present new research issues.

Zheng et al., (2002) introduce a new performance criterion, called caching efficiency, and propose a generic method for location-dependent cache invalidation strategies. In addition, Zheng et al. proposed two cache replacement policies: probability area and probability area inverse distance.

Xu et al., (2003) proposed three schemes to handle the location-dependent cache invalidation issue: bit vector with compression, grouped bit vector with compression and implicit scope information.

Another related area is the semantic cache model. (Dar et al., 1996) were the first work to use semantic distance function for replacement policy. They utilize the Manhattan distance function to calculate the distance of each semantic region from the user's current location. Their algorithm discard regions according to the values computed.

(Ren and Dunham, 2000) extend the work of Dar et al. to investigate ways in which semantic caching can be used to manage location-dependent data. Their work includes a formal model to represent moving objects and propose strategies of applying semantic caching to location dependent applications. For cache replacement, Ren and Dunham (2000) propose a semantic cache replacement policy called FAR (Furthest Away Replacement).

Similar to ours, other works such as (Ren and Dunham, 2000) uses the semantic caching model to manage location-dependent data. However, our cache management strategies are based on the geographical relationship between the query and the semantic segments. Our semantic cache model group in the same semantic segment, results of queries with related predicate and also spatial properties. The spatial information is the minimum bounding rectangle that represents the valid spatial scope of a semantic segment.

## 3 LOCATION-DEPENDENT SEMANTIC CACHING MODEL

Different query types require different indexing and query-processing strategies. Initially, this model supports *window query* (Gaede and Graefe, 1998).

Given a rectangle d-dimensional $R \subseteq E^d$, a window query searches all objects whose geographic position is inserted in the rectangle. The window extension is determinate trough the mobile client current location and the distance expressed in the query predicate. The supported operator is selection on single relations.

The following definition describes the location dependent query supported by cache.

**Definition 1 (Location Dependent Query)** *Given a database $D = \{R_i\}$, a location dependent query Q is a tuple ($Q_R$, $Q_P$, $Q_J$, $Q_L$, $Q_C$) where $Q_R \in D$ represents the relation issued, $Q_P$ a location predicate, $Q_J$ the window query, $Q_L$ the mobile client current location and $Q_C$ represents the query results.*

From now on, we also refer to location dependent query as the single word query.

### 3.1 Model Organization

The following definition describes the semantic segment.

**Definition 2 (Semantic segment)** *Given a relation R and its attributes set $A_1$, $A_2 \ldots A_n$, a semantic segment S on relation R is a tuple ($S_{ID}$, $S_R$, $S_P$, $S_A$, $S_C$, $S_{TS}$, $S_G$) where $S_{ID}$ stands for the segment identifier, $S_R$ the relation issued by the query, $S_P$ the select condition, $S_A$ the valid spatial scope of the*

*segment, $S_C$ the first page address of the segment content, $S_{TS}$ a timestamp and $S_G$ the segment group.*

To maintain the spatial information $S_A$, we use the minimum bounding rectangle (MBR) (Ix = [$x_1$, $x_2$] and Iy = [$y_1$, $y_2$]), that represents the geographic area of all data in the segment. MBR is a simple and efficient method to store geographic scopes. The semantic cache index is illustrated through the following example.

**Example 1**: Consider an object-relational database in the server with two relations: Restaurant (id, name, type, geometry), Hotel (id, name, price, geometry). The mobile unite position is represented by $MU_P$ = (x, y) and the following queries $Q_1$ and $Q_2$ are issued on time $T_1$ and $T_2$ respectively:

$Q_1$ $UM_P$ (10, 20): "Give me all hotels within 5 miles with price lower than U$ 100".

$Q_2$ $UM_P$ (20, -20): "Give me all Chinese restaurants within 10 miles".

Suppose that the client cache can't contribute with the answer. Thus, these queries are sent to the server and the results are stored in cache. The semantic cache index is shown as Table 1. Next section addresses the query processing over semantic cache scheme.

## 3.2 Semantic Query Processing

To perform a query, it is checked whether the query can be answered by the cache. If the query result can be totally obtained from the cache content, there is no communication with the server.

When the query can only be partially answered, there are two disjoint portions: one that can be answered in cache (probe query) and another that requires data to be fetched from the server (remainder query). The procedure of query partition is named query trimming (Ren et al., 2003). We define the probe and reminder query as follows.

**Definition 3 (Probe Query)** *Given a query Q on a relation R, S={$S_i$} denotes the set of semantic segments in the client cache that contributes with the query answer, where $S_{iR}=Q_R$, $S_{iA} \cap Q_j <> 0$ and $Q_P \cap S_P <> \varnothing$. The probe query is the union of all partial results obtained from each segment in S.*

**Definition 4 (Reminder Query)** *Given a query Q on a relation R. The remainder query (RQ) retrieves from the server the portion of Q not found in cache, that is, RQ = Q – PQ.*

According to the definitions above, PQ $\wedge$ RQ = false and PQ $\vee$ RQ = Q.

That is to say, the probe and the remainder query do not overlap and trough the union of the answers to these two queries we obtain the complete answer.

Our query processing model involves two steps: 1) to select the semantic segments candidate set; 2) to process the query against each candidate segment and the database in the server when is necessary. We present the algorithm 1, which selects the set of semantic segments (CjSC) that are candidate to answer the probe query.

```
Algorithm CjSC (Q, SC)
    Input: Query Q, Semantic Cache SC
    Output: Semantic segments candidates set (CjSC)
    {  Cj ← null
       for each relation Ri in the index structure do {
          if (Ri = QR) then
                    Cj ← segments Si…Sn }
       for each segment in Cj do {
          if  intersection (SAi , QJ) = null then
                Cj ← Cj -  Si  }
       for each segment in Cj do {
          if  (QP ⇒ SP) or (QP ∧ SP) then
                CjSC ← Si }     }
```

Algorithm 1: Semantic segments candidate set selection

Given a query Q and a semantic cache SC, firstly the algorithm selects the semantic segments that have the same relation of Q, that is, $S_R = Q_R$. Secondly, for each selected segment, its geographic area is compared with the window query. If the semantic segment that doesn't have any relationship with the window query it is eliminated from the set. Lastly, our algorithm analyzes the predicate similarity. At the end, we have the semantic segments that probably contribute to the answer. We illustrate this procedure with the example2.

**Example 2**: Consider that Q is issued by $UM_P$ (15,45): "Give me all hotels within 5 miles and diary price lower than U$ 100" (figure 1). According to the algorithm 1, Cj = {$S_1$, $S_3$, $S_6$, $S_7$}. The segment $S_6$ is eliminated form Cj because its intersection with $Q_j$ is null. Finally, the segments from Cj whose predicate satisfies the query predicate are added in CjSC set.

To avoid false segments selection, it is possible to use others geometric forms, such as circles or complex polygons. However, such representations consume a large portion of space and they introduce additional complexity to determine intersection areas.

Table 1: Example of Semantic Cache Index

| $S_{ID}$ | $S_R$ | $S_P$ | $S_A$ | $S_C$ | $S_s$ | $S_G$ |
|---|---|---|---|---|---|---|
| $S_1$ | Hotel | price<100 | [(5,15), (15,25)] | 4 | $T_1$ | 1 |
| $S_2$ | Restaurant | type = "Chinese" | [(10,-30), (30,-10)] | 8 | $T_2$ | 1 |

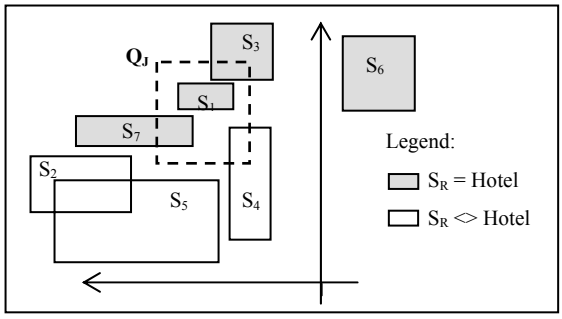


Figure 1: Example 2, semantic segments candidate

Next, we present the algorithm 2 which processes the query against the semantic segments in cache and the database if necessary.

**Algorithm QueryProcess (Q, CjSC )**
Input: Q e CjSC
Output: $Q_R$
{ For each $S_i$ in CjSC do {
$I_i \leftarrow$ intersection ($S_{Ai}$, $Q_J$)
If $Q_P \cap S_{iP} = Q_{iP}$ then { //** $Q_P \wedge \neg S_{iP}=0$
Execute $Q_i$ in $S_i$ and area $I_i$
$PQ_R \leftarrow PQ_R \cup Q_{iR}$
$X \leftarrow X + I_i$
ReorgSeg (Q, $I_i$, $S_i$) }
Else ($Q_P \cap S_{iP} \neq Q_{iP}$) {
$AQS_{iP} \leftarrow (Q_P \wedge \neg S_{iP})$ and area $I_i$
Send to server $AQS_i$
$PQ_R \leftarrow PQ_R \cup AQS_{iR}$
Execute $Q_i$ in $S_i$ and area $I_i$
$PQ_R \leftarrow PQ_R \cup Q_{iR}$
$X \leftarrow X + I_i$
ReorgSeg (Q, $I_i$, $S_i$) } }
If X <> $Q_J$ then send to server $RQ = Q \wedge \neg X$
$Q_R = RQ_R \cup PQ_R$
Create a new semantic for $Q_R$ }

Algorithm 2: Query processing in segments candidate set

For each segment in CjSC the following steps are applied. First, is defined the intersection area between the query window ($Q_J$) and the segment geographic area ($S_{iA}$). Following, the algorithm verifies the relationship between the segment ($S_{iP}$) and the query predicate ($Q_P$). If the segment $S_i$ satisfies the query, it is not required data from the server. The query is executed over $S_i$ and its results contribute with the probe query.

When a semantic segment $S_i$ answer partially the query, a partial sub-query named *amending query* is sent to the server. This amending query issues only the predicate that is not in $S_i$.

The intersection area $I_i$ that it has already researched in cache is stored in the vector X. Last of all, if X <> $Q_J$, the algorithm executes the reminder query in the server. The reminder query, represented by $Q_J \wedge \neg X$, returns all tuple that are inside the window query and that are not in X. We illustrate the algorithm 2 with the following example.

**Example 3**: With the query described in Example 2, suppose that $S_1$ and $S_3$ contribute with the query answer. Assume that the segment $S_3$ stores hotels with price lower than U$ 50 and the segment $S_1$ stores hotels with any price. In this case, $S_{3P}$ satisfies partially $Q_P$ and $S_{1P}$ satisfies totally $Q_P$.

Note that for the segment $S_3$, all its contents contribute to the result of Q and the remaining data (hotel with price between U$ 50 and U$100) are requested to the server (amending query $AQS_3$). Otherwise, $S_1$ has the complete predicate to answer Q and the amending query is not required.

We use the vector X that stores the rectangle of the areas already searched in cache. In Example 3, the reminder query is executed as follows: "SELECT * FROM Hotel WHERE Hotel.price < 100 AND ((Hotel.geometry IN $Q_J$) AND (Hotel.geometry NOT IN X))".

## 3.3 Semantic Segment Reorganization

The previous sections described our SC model as well as the query processing. In this section, we deal with the scenario after query processing and semantic segment reorganization.

To avoid redundant data, we cannot keep both S and Q in the cache. When a query is executed, if it is completely answered by the server, the complete answer is stored in a new segment. However, when a portion of the query answer is already in cache, only the data brought into the cache as the result of the remainder query should be stored in a new semantic segment. Therefore, the semantic segments can be modified dynamically based on the queries that are executed at the client.

The way that semantic segments are formed has a significant effect on the performance of the semantic cache (Jónsson, 1998). In addiction, this model considers the geographic relationship between Q and S to reorganize the semantic segments. The algorithm 3 executes the segment reorganization.

Procedure ReorgSeg (Q, $I_i$, $S_i$)
Input: Q, $I_i$ and $S_i$
{ If $I_i$ < > $S_{Ai}$ then {
$S_{Ai} \leftarrow S_{Ai} - I_i$
If $S_{Ai}$ < > rectangle form then {
Adjust $S_{iA}$ with a rectangle representation } }
$S_i \leftarrow S_i - ( (S_i \cap Q)$ in $I_i \wedge (S_i \wedge \neg Q)$ in $I_i )$
If $S_i \wedge \neg Q$ < > null then {
Create a new segment $S_i$'
$S_i' \leftarrow (S_i \wedge \neg Q)$ in $I_i$ } }

Algorithm 3: Semantic segment reorganization procedure

Given a query Q, a semantic segment $S_i$ and the intersection area between Q and $S_i$ named $I_i$, the segment reorganization is given as follows. Firstly, the procedure verifies if $I_i$ is different from $S_{Ai}$. When a window query $Q_j$ covers totally $S_A$ (figure 1 - segment $S_1$), it is not necessary to actualize $S_{Ai}$ because the intersection area $I_i$ is the same of $S_i$.

On the other hand, when $S_{Ai} \cap Q_j$ is different from $S_{Ai}$ (figure 1 - segment $S_3$), it is necessary to reduce from $S_{Ai}$ the intersection area $I_i$. If the remaining geographic area has a complex form, it must be adjusted as a rectangle shape.

After geographic area verification, the semantic segment content should be updated. The data used to answer Q must be removed from S, and if it remains data, these remaining data should form a new semantic segment $S_i$'. The complete result of Q always will generate a new segment. We illustrate this procedure with the following example.

**Example 4**: Consider the query "Give me all hotels within 5 miles with price higher than U$ 100". Suppose that the segment $S_3$ stores hotels with diary price higher than U$ 50. In this case, $S_{3P}$ satisfies $Q_P$ and also $S_{3A}$ intersects partially with $Q_J$. After query execution in $S_3$, this segment must be reorganized (Figure 2).

Note that $S_{A3} \cap Q_J$ is different from $S_{A3}$, then $S_{A3} \leftarrow S_{A3} - I_3$. The resulting area has a complex form, and then it is adjusted as a rectangle shape. The data used to answer Q ($S_3 \cap Q$) must be removed from $S_3$ (that is, Hotel.price > 100 in $I_3$). The remaining data $S_3 \wedge \neg Q$ (that is, Hotel.price > 50 and Hotel.price < 100 in $I_3$) is not null, then a new segment $S_3$' is created with these remaining content.

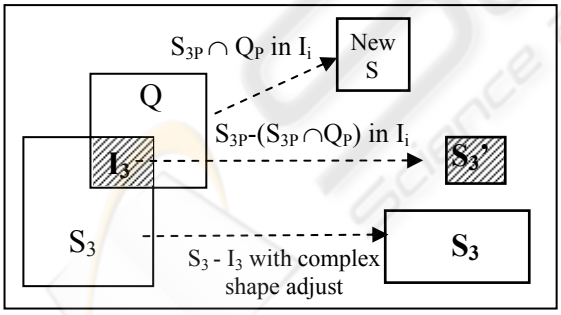


Figure 2: An example of Segment Reorganization

# 4 LOCATION DEPENDENT CACHE ARCHITECTURE

In this section we propose a Location Dependent Semantic Cache Management – LDSCM (figure 3), an architecture that support LDQ and provides semantic cache management based on spatial property. In the server, placed at the fixed network, an object-relational database system stores and manages the LDD. The mobile clients communicate to the server through a wireless link and are responsible for storing and managing its semantic cache. Next section details the LDSCM components.

The component LDQ Query Generator generates location dependent queries according to the database scheme in the server. This module is responsible for: (1) to elaborate a query; (2) to bind the user current location and to send the query to the component LDQ Query Processor; (3) to receive the complete answer from LDQ Query Processor and to send it to the application.

Initially, an application won't be represented for user interface because the main objective of this prototype is to test the proposed solutions. Once proved the benefits of this new model, this will be able to be implemented according to the characteristics of different mobile devices, such as notebook, palm, PDA and so on.

The component LDQ Query Processor is responsible for the following tasks: (1) to determine and execute the probe query, amending query and reminder query, according to relationship with the semantic segments in cache; and (2) to combine the results obtained in cache with the results shipped from the server and return the complete answer to the component LDQ Query Generator. To accomplish these tasks, the LDQ Query Processor is composed by the following modules: Segments Candidates Selector and Query Divisor and Executor. The module Segments Candidates Selector is responsible for determining the set (CjSC) of semantic segments that are candidates to answer the query Q. This process is executed using the algorithm 1, as described in section 3.

Once defined the CjSC, the Query Divisor and Executor component defines and execute the probe query for each segment in CjSC as shown in algorithm 2. This module also sends to the server the remaining and amending queries when necessary. The server receives queries from the client, processes them, and sends the results back.

The complete probe query is composed by one or more sub-queries. The reminder query result is combined with the probe query to compose the complete answer.

The portion of the answer that is not in cache (reminder query) must be sent to the component Cache Structures Manager. This component reorganizes the semantic segments, as described in algorithm 3.

The Cache Structures Manager is also responsible for semantic caching replacement. If necessary, the replacement is performed using the ASCR policy (Manica et al., 2004).

# 5 CONCLUSIONS AND FUTURE WORK

Semantic caching is a new client caching architecture, which focuses on reducing network traffic and improving data availability in case of disconnection.

We proposed a new semantic caching model for location dependent information systems. The characteristics of the SC architecture, SC query processing strategy and practical issues of client caching management were described. One of the main contributions of this work is the use of geographic information on cache management.

Here we describe the design of our prototype, the next step is to study its performance and compare with other proposals.

Future studies also will explore semantic cache management issues for complex location-dependent queries.

# REFERENCES


Dar, S., Franklin, M. J., Jónsson, B. T., Srivastava, D., and Tan, M., 1996. Semantic Data Caching and Replacement. In: *Proceedings of the 22th International Conference on Very Large Data Bases*, Mumbai (Bombay), India, pp. 330-341.

Gaede, V. and Graefe, V., 1998. Oversize Shelves: A Storage Management Technique for Large Spatial Data Objects. *International Journal of Geographical Information Science* (IJGIS), 11 (1): 5-32.

Jónsson, B. T., 1998. Application-Oriented Buffering and Caching Techniques. Dissertation submitted to the Graduate School of the University of Mayland.

Lee, D. L., Xu, J., Zheng, B., and Lee, W., 2002. Data Management in Location-Dependent Information Services, *IEEE Pervasive Computing*. **1**(3):65-72.

Lee, K. C. K., Leong, H. V., and Si, A., 1999. Semantic Query Caching in a Mobile Environment, *ACM Mobile Computing and Communications Review*. **3**(2):28-36.

Manica, H., Camargo, M. S., Ciferri, R. R., and Ciferri, C. D. A, 2004. A New Model for Location-Dependent Semantic Cache Based on Pre-Defined Regions. In: *Proceedings of the 30th Conferencia Latinoamericana de Informática*, Arequipa, Peru, pp. 533-542.

Manica, H., Camargo, M. S., Ciferri, R. R., and Ciferri, C. D. A, 2004-b. Processamento de Consultas Espaciais Baseado em Cache Semântico. In: *Proceedings of the 6th Brazilian Symposium on GeoInformatics*, Campos do Jordão, Brazil, pp. 423-435.

Ren, Q., and Dunham, M. H., 2000. Using Semantic Caching to Manage Location Dependent Data in Mobile Computing. In: *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, Boston, MA, USA, pp. 210-221.

Ren, Q., Dunham, M. H., and Kumar, V., 2003. Semantic Caching and Query Processing, *IEEE Transactions on Knowledge and Data Engineering*. **15**(1):192-210.

Xu, J., Tang, X., and Lee, D. L., 2003. Performance Analysis of Location-Dependent Cache Invalidation Schemes for Mobile Environments, *IEEE Transactions on Knowledge and Data Engineering*. **15**(2):474-488.

Zheng, B., Xu, J., and Lee, D. L., 2002. Cache Invalidation and Replacement Strategies for Location-Dependent Data in Mobile Environments, *IEEE Transactions on Computers*, Special Issue on Database Management and Mobile Computing. **51**(10):1141-1153.


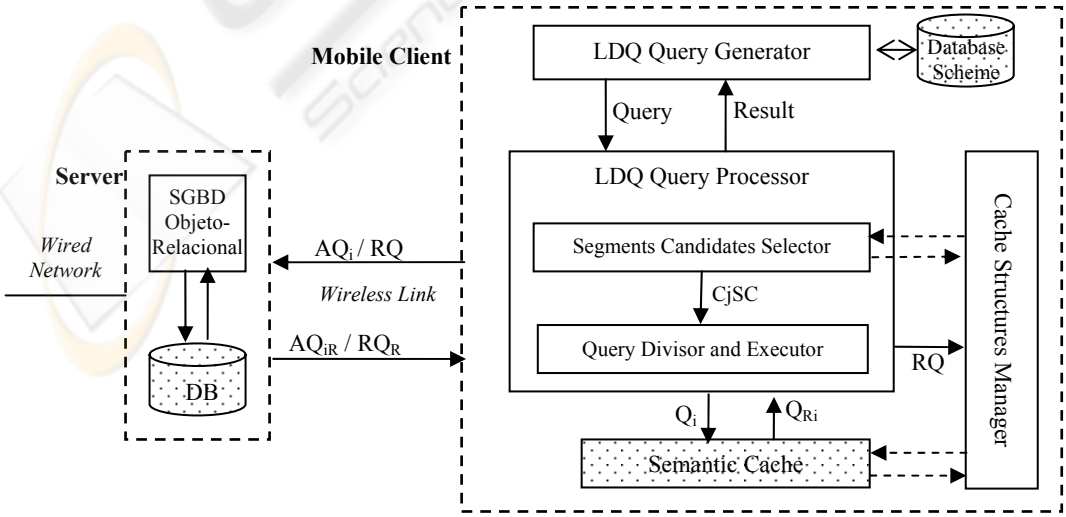


Figure 3: LDSCM Architecture