PROFILE-BASED LOCATION-AWARE INFORMATION RETRIEVAL FOR MOBILE CLIENTS

The members of the Committee approve the master's thesis of Anupama Mutt

Sharma Chakravarthy
Supervising Professor _____

Mohan Kumar
Supervising Professor _____

Hao Che _____

To my husband, parents, family and friends.

PROFILE-BASED LOCATION-AWARE INFORMATION RETRIEVAL FOR MOBILE CLIENTS

by

Anupama Mutt

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2004

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest sincere gratitude to my advisors, Prof. Sharma Chakravarthy and Prof. Mohan Kumar, for giving me an opportunity to work on this challenging problem and providing me great guidance and support through the course of this research work. I would also like to thank Prof. Hao Che for serving on my committee.

I am also grateful to Shravan Chamakura and Manu Aery for maintaining a well-administered research environment and their commitment to work. Sincere appreciation is due to Raman Adaikkalavan, Ajay Eppili, Suchitha Joseph and all friends in ITLAB for their invaluable help and advice during the course of development of this system. This work was supported, in part, by NSF (grants IIS-0123730 and IIS-0326505).

Last, but not the least, I would also like to thank my husband, parents and family members for their endless love and constant support throughout my academic career. Without their encouragement and endurance, this work would not have been possible.

November 19, 2004

ABSTRACT

PROFILE-BASED LOCATION-AWARE INFORMATION RETRIEVAL FOR MOBILE CLIENTS

Publication No. _____

Anupama Mutt, M.S.

The University of Texas at Arlington, 2004

Supervising Professor: Sharma Chakravarthy and Mohan Kumar

Advances in wireless and mobile computing allow mobile clients to perform a wide range of applications. In certain applications, mobile clients may be interested in getting information periodically. For example, a user may need weather information every hour. This thesis addresses the problem of computing profiles (queries) efficiently and the caching of results based on the location of clients as they move. To this end, we propose a scheme that offers personalized, location-aware information access in mobile environments.

In the proposed scheme, the nodes on the fixed network execute and manage profiles on behalf of mobile clients. A simple but effective profile specification language to capture user intent has been used for retrieving information from disparate sources such as WWW and relational databases. The results for the profiles are computed and cached

at appropriate nodes (based on the location of clients) and pushed to the user in a timely manner according to user specifications. The execution of the profiles is maintained on a node close to the mobile client in order to reduce the communication cost of result notification. In the proposed scheme, profiles and their executions are migrated to minimize data transmission costs. Simulation results show that the proposed scheme performs better than the existing approaches when bandwidth usage and percentage data loss are considered.

Active capability (ECA rules) has been adapted to process profiles and communicate notifications efficiently. Several mobile clients submit profiles to access location-aware information. As the execution of the profiles is done close to the location of the mobile clients, there will be considerable number of similar profiles that require processing on the same node in the fixed network. The proposed scheme groups similar queries to share common computation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Mobile devices such as cellular phones and personal digital assistants are widely popular because of their size, portability and their ability to allow a mobile user to perform a wide range of applications once limited to wired computing environments [1]. Such applications include accessing data, performing computation, etc., using mobile devices. However, the limited bandwidth and high latency of today's wireless networks greatly inhibit supporting such applications over wireless network. In addition, mobile devices are constrained in terms of CPU, power, memory, disk, display capabilities etc [2].

In the cellular network infrastructure the whole geographical area is divided into cells. Each cell has a Base Station (BS) that communicates with the mobile clients in its cell area through a wireless channel. The BS is also referred to as a server and the mobile client as a mobile station or a mobile node. BSs serving an area are connected by a wired network. When a mobile client moves from one cell to another, the wireless link with the old BS is broken and a new link is established with another BS.

Mobile clients need to access information from various applications. This includes access to legacy and existing applications on the wired network with minimum effort and cost. Mobile clients also need access to information anytime, anywhere. For example a mobile user would like to know current weather conditions when he/she is out on a camping trip. The challenge is to support access to disparate data sources such as World Wide Web (WWW) and databases that can provide useful information to mobile clients, without burdening the mobile device in terms of computation and power.

Mobile clients are typically connected over wireless channels. Wireless networks provide lower bandwidth than the wired networks. Data can be preprocessed, filtered or adapted to the needs, preferences and characteristics specified by the mobile user to reduce the bandwidth usage over the wireless link. The issues of accessing data from various disparate sources and customizing them to the specifications of the mobile user need to be addressed.

Queries given by mobile user have different characteristics. Based on characteristics, they can be traditional, location-dependent or periodic queries. Processing of location-dependent queries for a mobile user become complex because of the frequent change in the location of the mobile user. A query including any location related attribute in its predicates is defined as location-dependent query [3]. For example, a user might want to know weather information in his/her location. Location-dependent queries pose a new kind of requirement on the query processing subsystem as the location predicate in the query changes dynamically. On the other hand, periodic queries are queries that require periodic computation of results. They are nothing but traditional queries or location-dependent queries with periodicity. For example, "find the weather information every hour". To execute periodic queries, query processing subsystems need to be able to compute results periodically, maintain intermediate information for a long duration and push results to the mobile client in a timely manner without the user having to make repeated requests. There has been considerable amount of work done for processing traditional queries [4, 5] and location-dependent queries [3] in mobile environments, but issues of processing periodic queries have not been addressed.

Clearly, having the query processing subsystem on the mobile device is not an option considering the amount of computation and power that is required to process queries posed by mobile clients [6]. Client-proxy-server model is the general approach followed by the current systems to process queries in the wired network. In this approach, queries

are processed on a proxy (intermediary wired node) that mediates between primary information sources and mobile clients. Systems adopting client-proxy-server approach can be either centralized or distributed. Most of the current systems are based on centralized proxies [5, 7, 8, 9]. Centralized systems do not scale very well as the number of mobile clients increase. Distributed systems like MobileIQ [4], eRace [10] have obvious advantages as they support load distribution to multiple processors, scalability of performance and improves system robustness and availability.

Given the importance of ubiquitous access to the information infrastructure, there is a need to develop techniques that alleviate the problems outlined above and seek to provide mobile clients the same degree of responsiveness and performance as a wired node. In particular, this thesis looks into the issues of providing personalized information access in mobile environments. We try to reduce the effects of low bandwidth and address problems that arise due to power and computation constraints of mobile devices. This thesis proposes a scheme called "InfoAccess" that offers personalized, location-aware information access in mobile environments using a distributed approach. In InfoAccess, user intent is captured in the form of profiles. Profile cache is a collection of profiles and cached results pertaining to a mobile client. The profile cache is maintained on a wired node close to the location of the mobile client. This reduces transmission latency of notifications to the mobile client while relieving the wired network of unnecessary network load that would have otherwise resulted from transmitting large volumes of notifications for long duration.

The main features of InfoAccess are:

1. A simple and effective profile specification language has been proposed to specify user queries.

2. Profile cache is dynamically moved from one BS to another to follow the movement of the user based on data transmission costs, thus making the profile cache location-aware.
3. Profile processing and result notification are done efficiently through the use of query graphs.
4. Active capability (ECA rules) [11, 12] has been explored to support periodic queries along with traditional and location-dependent queries.
5. As there can be similar queries that need to be processed on the same node on the fixed network, grouping has been exploited to avoid redundant computation.

The thesis is organized as follows. Chapter 2 introduces profiles and profile specification language that is used to specify user requests. Chapter 3 gives an overview of the architecture and discusses the functionalities of various modules of the InfoAccess server. Chapter 4 describes the query graphs and explains how active capability and query graphs are used to compute results and notify the mobile client efficiently. Chapter 5 presents the metrics based on which the profile cache is moved dynamically. Chapter 6 presents simulation results and discusses implementation issues. Chapter 7 reviews related work. Finally, Chapter 8 outlines conclusion and future work.

CHAPTER 2

PROFILE SPECIFICATION

In this chapter, we investigate the use of user profiles to provide information that is desired by the user in an unobtrusive manner. Requested information is fetched according to the specifications in the profile. Once the information is fetched, effective and timely propagation of information to the user is important. Results need to be notified to users in different ways based on their preferences, the type of mobile device they use etc. Hence, there is a need to define a simple and expressive specification language using which the user can specify the data required. The user should also be able to specify predicates (filter conditions) on the fetched data, if any. The user should be able to tell the system how and when the results should be notified. This chapter describes the profile specification language that can be used to specify interests and preferences of the user in the form of profiles.

2.1 Profiles

In InfoAccess, profiles give a concise description of user interests. They represent a set of traditional queries or location-dependent queries or long-term periodically evaluated queries. These queries can be typical queries to fetch data from web sources, or relational databases. A profile has both data parameters and control parameters. Data parameters are query arguments (today's temperature on http://www.weather.com), whereas control parameters are the ones that determine the time at which query execution should be started, specify the frequency of query execution and notification. The profile is specified using a simple and expressive profile specification language. Each user can have a

```
<Profile> :=  Create-Profile<profile-name>
              [Lifespan <lifespan -range>]
              Data required <basic -functionalities> | Source <website> |
                                Database<database, tables, userId, password>
              [Keywords <w ords>]
              [Filter Conditions <expression>]
              [Location <none| addr>
              [Notify {on -change| once | hourly | daily| monthly}]
              [Notification Device {laptop| PDA | mobile phone}]
              [QoS Freshness of Data <time>]

<profile-name> := Identifier
<lifespan-range> := short | long | medium
<basic-functionalities> := weather info
                           | Restaurants [<addr>]
                           | Post Offices [<addr>
                           | Gas stations [<addr>]
                           | Local events [<addr>]
                           | Lodges [<addr>]
                           | Maps {[<addr1>] to <addr2>}
<website> := url
<words> := <word1> [[[[ ,word2] ,word3] …. wordn]]
<expression> := <expr> [<logical op> <expr> [<logical op> <expr>……..]]
<expr> :=  <attribute><relational op><value>
<logical op>  := AND | OR
<relational op> := > | >= | < | <= | = = | !=
<attribute> := <word>
<value> := <integer>
<addr> := <city,state>|<zipcode>
<time> := <integer> {seconds | minutes | hours}
```

Figure 2.1 Profile Specification Language

number of profiles that are submitted to the InfoAccess servers. The servers process the requests according to specifications given in the profile at the right intervals of time and push results to the user. This reduces the burden on the mobile user as he/she has to submit a profile only once and the required data will be made available to the user just in time over the lifespan of the profile.

2.2 Profile Specification Language

Profile specification language provides a well-defined semantics for specifying user interests and preferences. The semantics of the profile specification language should be simple and straight forward for the user to populate. At the same time, the language should be descriptive enough for the system (InfoAccess) to be able to interpret it and formulate valid queries from the specification. The formal syntax of the profile specification language is as shown in figure 2.1

2.2.1 Profile-name

The system generates a unique identifier to keep track of the profiles for a mobile client. In addition, the system allows the user to specify a name of his/her choice. This helps the user in keeping track of profiles. The user can enable or disable the profile or delete the profile at his/her wish using the profile-name.

2.2.2 Lifespan-range

Lifespan-range is the time period during which the profile will be executed. The user can specify long, medium, short lifespan or time period for the lifespan. A profile with long lifespan is executed for a month, medium is executed for a week and short is executed for a day. The default is medium lifespan.

2.2.3 Data Required

Data required clause allows the user to specify the data sources from which the required data needs to be fetched. The user can specify a webpage or a relational database as a source. He/She can also choose to get data from one of the basic functionalities. If the user specifies database as a source, additional information such as names of tables, userId and password are collected.

Basic functionalities can be weather information, restaurants, post offices, gas stations, local events or lodges. These are some local information that many mobile users might require. If the user selects one of the basic functionalities, he/she does not have to provide the data source (webpage or a database name). The system knows the data source for these functionalities and can provide results for the user. This reduces burden on a user who is looking for local information in an unknown place. Based on the information provided in this clause, appropriate requests are formed and sent to the data sources to fetch the data.

2.2.4 Keywords

Keywords are the words that are of interest to the user. There can be a lot of information in a webpage or a database table that is not of significance to the user. If the user is looking for some specific information, it should be specified as a keyword. For a database, project attributes are specified as keywords. For a webpage, words that are of interest to the user are specified as keywords. For example, user could specify temperature in the www.weather.com webpage. The system will scan the page to fetch the current temperature and send it out to the user instead of sending the whole page.

2.2.5 Filter Conditions

Filter conditions are the predicates that should be applied to the data. They filter the data and reduce the amount of data that needs to be transmitted to the user. For a database, predicates in the where clause of the sql query are specified as filter conditions. For a webpage, filter conditions can be specified only if the resultant data is a scalar quantity. For example, if the user specifies, "temperature $>$ 90" as a filter condition, then the temperature is notified to the user only if temperature $>$ 90.

2.2.6 Location

Location clause allows the user to specify his/her location. This in turn allows the user to specify location-dependent queries. The user can specify either city, state or zipcode as a location attribute. The default for the location attribute is the current location of the user. The current location of the user is found by querying a location service, that can provide the location of a mobile client in terms of zipcode.

2.2.7 Notify

Notify clause is the one which allows the user to specify periodic queries as well as traditional queries. The user can specify that he/she needs to be notified every t time period. The user can specify once, hourly, daily, weekly or every n minutes/hours. This clause represents the time interval for which the data need to be fetched. The system fetches the data and computes the result and notifies the user every t time period. The notify clause is really helpful for keeping track of information that changes frequently. For example, a user can specify that temperature has to be notified every hour. The system computes temperature every hour and pushes it to the user.

2.2.8 Notification device

The user can specify his/her device. The system maintains the device profiles for each of the device types. Based on this, the system tailors the result according to the device capabilities. For example, if a particular device is not capable of displaying HTML pages, only text information is sent to the user.

2.2.9 QoS Freshness of data

This is a quality of service parameter, which denotes the amount of delay that the user can tolerate. It is the extent to which the user can tolerate stale data. It is

provided in units of time. For example, if the user has specified “5 minutes”, then the user is ready to accept data that are fetched no earlier than the last 5 minutes. This is an important parameter as this can be used to prefetch data such that the user receives results virtually with no latency.

2.2.10 Examples of profiles

On the whole, profile specification language provides the system a comprehensive set of information to understand the user intent and formulate meaningful queries. Also, from a user’s perspective, this profile definition will be easy and intuitive to describe. Following are some scenarios and their representation using profile specification language.

Scenario 1: A user is on a camping trip. Thunderstorms are forecast for the night. He/she wants to get back home before it starts to rain. The user wants to know the temperature every hour if the temperature falls below 50 F. The profile for the above scenario is:

Create-profile P1

Data-Required Weather

Keywords Temperature

Filter Condition Temperature $<$ 50

Notify every 1 hour

Notification Device PDA

Notification Mode pop-up window

QoS Freshness of data 10 min

Scenario 2: A restaurant database has tables called “location” and “cuisine”. The location table has the attributes ID, restaurantName and zipcode. The cusine table has the attributes ID and cuisineType. A user wants to find the restaurants in the zipcode “76011” that serve american cuisine. The user is aware of the restaurant database and

he/she has privileges to access data from this database. The profile for this scenario is as follows:

Create-Profile P1

Lifespan Medium

Data required Database: Restaurant

Tables: location, cuisine

UserId: AAAA

Password: PPPP

Keywords restaurantName

Filter Conditions location.ID=cuisine.ID AND cuisineType="American"

Location 76011

Notify once

Notification device PDA

2.3 Summary

The user submits profiles that are specified using the profile specification language. The profiles are executed on the wired network according to the specification of the user and the results are notified.

CHAPTER 3

ARCHITECTURE OF INFOACCESS

InfoAccess aims at providing personalized, location-aware information to subscribed users. InfoAccess is a proxy-based service that is implemented as a distributed system. It consists of a network of InfoAccess servers that are deployed on base stations on the wired network. InfoAccess servers accept queries in the form of profiles. InfoAccess servers work together to process profiles and push customized results to mobile clients. This chapter explains in detail the architecture of the InfoAccess server, various modules of the InfoAccess server and how they work together to efficiently process profiles and deliver results to mobile clients.

3.1 Requirements

Information retrieval for mobile clients has the following requirements.

- The system should provide services, that offer personalization, customization and localization of data.
- The system should optimize the useful content that reaches users.
- Mobile devices with varying capabilities need to be supported.
- The system should support push and pull based information retrieval.
- The system should support transparent access to disparate data sources such as WWW and the relational databases.
- The system should support periodic queries along with traditional and location-dependent queries.

3.2 Approaches for Handling Information Retrieval for Mobile Clients

Requirements mentioned in the previous section can be addressed by following various different approaches. Each of the following approaches is analyzed to find the best approach that suits the needs of InfoAccess.

3.2.1 End-to-End approach

In the end-to-end approach, data servers (information sources) adapt their content on-the-fly taking into account user profiles or the mobile device and the network connection involved in the communication [13]. However, for an end-to-end approach to work properly, adaptation software has to be inserted at each data server. Consequently, software updates have to be propagated to all data servers whenever new mobile devices or content-encoding protocols emerge. Moreover, on-the-fly adaptation of content can be very time-consuming leading to deterioration of user experience apart from overloading the data servers.

3.2.2 Client-Proxy-Server Approach

An alternative approach is the deployment of proxies. Proxies are software entities that are deployed between mobile clients and data servers on the wired network. They intervene the flow of information from data servers to mobile clients at the application level and modify the information according to user specifications. An important aspect that characterizes the client-proxy-server approach is whether it is composed of a centralized proxy running on one host, or a number of distributed proxies residing at different hosts that communicate among themselves to share the load.

3.2.2.1 Centralized Proxies Vs Distributed Proxies

Centralized proxies are deployed on a particular host on the wired network and a mobile client has to communicate with the same proxy for information retrieval [5, 7, 8, 9]. Mobile clients may find themselves far away from their proxies thus making the centralized approach expensive in terms of responsiveness and bandwidth usage. On the other hand, distributed proxy systems [4, 10] provide a number of proxies that are deployed all over the wired network. This permits the proxies to move the resources of a particular mobile user to follow the movement of the mobile user, which in turn reduces the response time for the mobile user and the bandwidth usage on the wired network.

Moreover, centralized proxies become overloaded when the number of mobile clients increase. Given the high-performance and scalability requirements for supporting the ever-growing number of mobile users, centralized proxies do not fit very well, thus necessitating the distribution of computation, storage and complexity in the wired network.

Distributed client-proxy-server approach is the best approach for providing information retrieval in InfoAccess for the above stated reasons. A number of distributed InfoAccess servers are deployed on the base stations of the mobile infrastructure. These servers mediate between the primary information sources and various mobile clients providing the functionalities discussed above.

3.3 Architecture of InfoAccess

The InfoAccess servers are deployed on all base stations on the wired network as shown in Figure 3.1. The mobile client communicates with the InfoAccess server that is close to its location. The mobile client basically submits profiles and receives results from the server. The InfoAccess server in turn schedules the profiles for execution, communicates with data servers, fetches the requested data at the specified intervals of

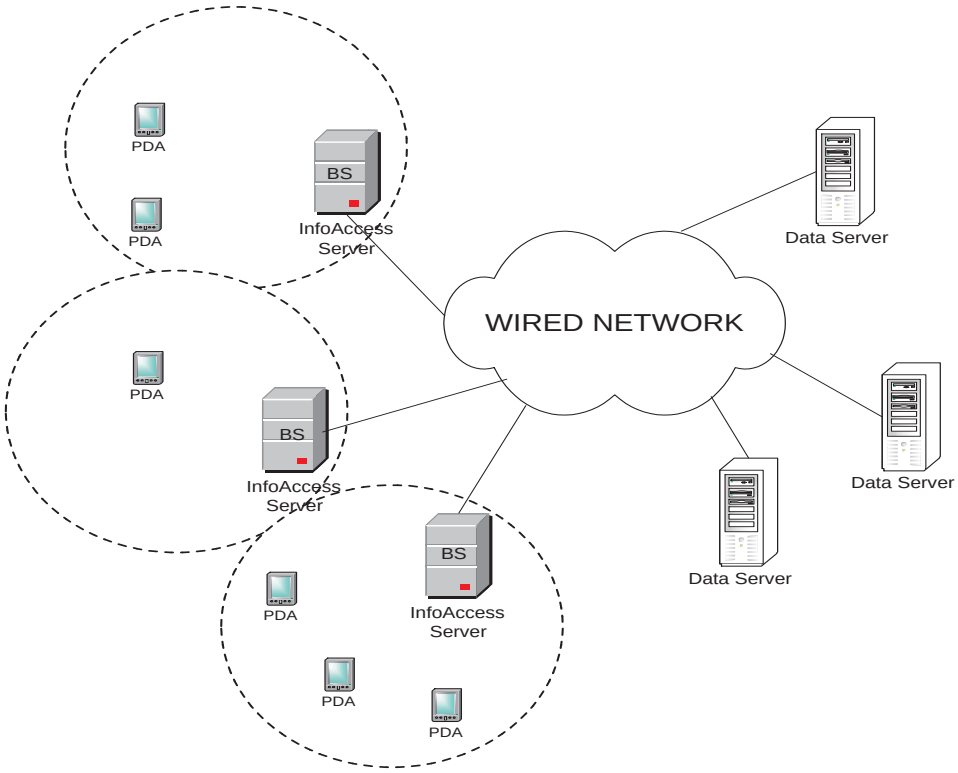


Figure 3.1 Overview of InfoAccess

time, customizes the data to the needs of the user as specified in the profile and pushes results to the user . Data servers are the primary information sources such as web servers and database servers.

3.3.1 Profile Cache

Profile cache is a collection of profiles and cached results for a mobile client. A profile cache is specific to a mobile client. Profile cache is maintained and managed by InfoAccess servers on the wired network. The most recent result of each profile is cached in the profile cache. The result is not sent to the mobile client across the wireless link if the result has not changed since the previous computation. This is done in order to

minimize the bandwidth usage across the wireless link at the cost of consuming extra memory space on the server.

Essentially, profile cache is maintained on the InfoAccess servers. In InfoAccess, profile cache is dynamically migrated from one base station to another to maintain the execution of profiles close to the mobile client. This not only reduces data transmission cost on the wired network but also reduces the communication latency for the mobile client. The InfoAccess servers communicate among themselves and decide where the profile execution for a mobile user should be done such that it benefits both the mobile client and the network load.

3.3.2 InfoAccess Server

InfoAccess server consists of two major components - Profile Manager and Profile-cache Manager as shown in the Figure 3.2. Conceptually, profile manager and profile cache manager are similar to home agent (HA) and foreign agent (FA) in the cellular networking model. Every mobile client registers itself with a base station, which acts as a HA for that particular mobile client. The HA for a mobile client remains the same even when the mobile client moves to a different base station. Home Agent always knows the location of the mobile client. In our architecture, profile manager for a mobile client is maintained on its home agent.

When the mobile client moves to a base station that is different from the home agent, it is serviced by a foreign agent. The address of the foreign agent is maintained as location information in the home agent. Both home agent and foreign agent work in conjunction to provide continuous access to the wired network for the mobile client. In our architecture, profile cache manager for a mobile client can be in the home agent or on one of the foreign agents.

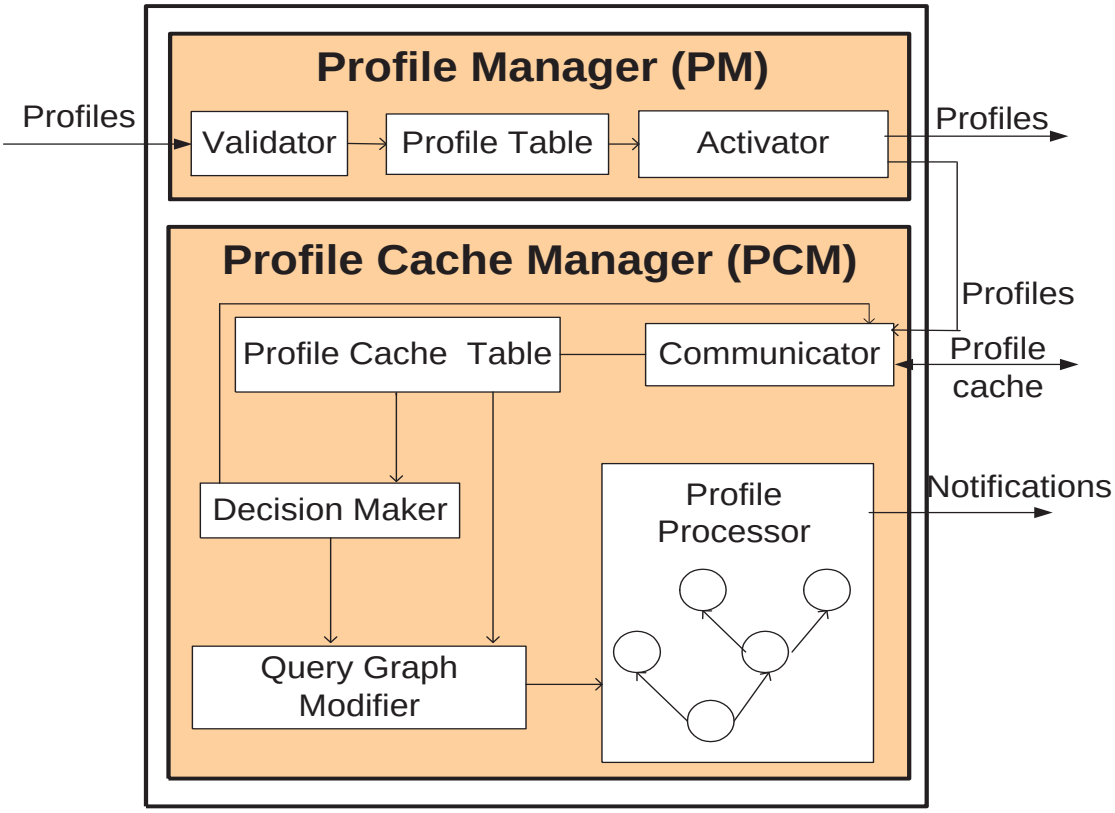


Figure 3.2 Architecture of InfoAccess Server

3.3.2.1 Profile Manager(PM)

The profile manager is responsible for storing all the profiles of mobile clients that are registered with this base station. Just as a home agent serves as the central point of contact to find the current location of the mobile client, profile manager serves as the central repository of profiles for a mobile client. At any given time, the profile manager is aware of the location (BS) of the mobile client and the location (BS) of the profile cache. Profiles that are scheduled to start executing at a later time are stored as dormant profiles. Profiles become active as soon as they start executing. The various tasks performed by the profile cache manager are summarized in the Figure 3.3

Tasks performed by the Profile Manager:

1.If the user inputs a new profile,
  i. Validate the profile
  ii. Store the profile in the profile table. Mark the profile as dormant.
  iii. Schedule the profile for activation at the start of the lifetime of the profile.
  iv. Schedule the profile for deactivation at the end of lifetime of the profile.

2. When the scheduled time for the activation of a profile elapses,
  i. Mark the profile as active
  ii. Send the profile to the profile cache manager that is currently executing the profile cache for this particular mobile client.

3. When the scheduled time for the deactivation of a profile elapses,
  i. Delete the profile from the profile table.

Figure 3.3 Pseudocode for Profile Manager

The profile manager has three modules: validator, profile table and scheduler as shown in the Figure 3.2. The validator accepts profiles from mobile clients, validates it and stores in the profile table. The profile table maintains the profile ID, the profile itself, the status of the profile(active or dormant), the location of the mobile client and the location of the proxy cache. As soon as the profile becomes active, the profile table updates the status of the profile to mark it active and notifies the scheduler. The scheduler schedules the profile for activation at the start of the lifetime of the profile. It also schedules for deactivation at the end of the lifetime of the profile. When the time for

activation elapses, the profile is marked active and sent to the profile cache manager that is currently managing the profile cache of the mobile client on a foreign agent. When the time for deactivation elapses, the profile is deleted from the profile table.

3.3.2.2 Profile Cache Manager(PCM)

Profile Cache Manager maintains and manages profile cache of mobile clients. It is responsible for building query graphs corresponding to profiles in the profile cache, scheduling execution of queries using active capability, customizing data to users specifications and notifying the results to the user. The various tasks performed by the profile cache manager are summarized in the Figure 3.4

The profile cache manager performs its functionalities with the help of various modules. The modules are shown in the Figure 3.2. The functionality of each module is explained below.

- Communicator - The communicator module receives profiles that are sent by profile managers of mobile clients. It sends and receives profile cache by communicating with other profile cache managers. It notifies the profile cache table of any profile that has been sent or received.
- Profile cache table - Profile cache table stores the metadata about all the profiles that are currently being executed by the profile cache manager. It maintains the profile ID, the profile itself, some state information like pointer to the query graph and the current location of the mobile client. Location of the mobile client is necessary because, in some situations, the profile cache manager can be executing profiles that has moved out of its coverage area into a different foreign agent. When this happens, the location is used to push the results to the user. When the profile cache table receives notification from the communicator, it adds the profile into the table and notifies the query graph modifier.

```
Tasks performed by the Profile Cache Manager:

1. If a profile is received from a profile manager,
   i. Store the information of the profile in the profile cache table.
   ii. Add the profile to the query graph and schedule the
       execution of the profile.

2. If the profile cache is received from another profile cache
   manager,
   i. Store all the profies in the profile cache into the profile cache
      table.
   ii. Add all the profiles in the profile cache to the query graph
       and schedule for execution of the profiles.

3. If the mobile client moves to a new BS
   i. Calculate the total cost factor for the profile  cache based on
      which decision maker decides whether to move the profile
      cache or not.
  ii. If the profile cache has to be moved,
      a) Delete all the profiles belonging to the profile cache from
         the query graph.
      b) Delete the profiles from the profile cache table.
      c) Send all the profiles belonging to the profile cache and the
         cached results to the new BS
 iii. If the profile cache is not moved,
      a) Note down the current address of the mobile client in the
         profile cache table.

4. For all the profiles in the query graph
   i. Start computation
  ii. Notify results to users as and when they are computed.
```

Figure 3.4 Pseudocode for Profile Cache Manager

- Query graph modifier - A query graph corresponding to each profile is built in order to compute the results efficiently. They provide a means of combining the computation of partially or exactly similar profiles. Query graph modifier converts the profiles into query graphs. Query graph modifier basically builds the query graph, adds profiles to existing graph if they are similar, deletes profiles from the graphs when the profile cache moves to a different base station or the lifetime of the profile has expired.

- Profile processor - Profile processor is the module that executes queries in the query graphs, computes results and notifies the user at specified intervals of time.
- Decision Maker - When a mobile client moves from one base station to another, the decision module makes a decision on whether to move the profile cache or not based on data transmission cost. If it decides to move the profile cache, it gets the profiles belonging to the profile cache and their recent results, then informs the query graph modifier to delete those profiles, and notifies the communicator to move the profile cache across the network to the profile cache manager on the new base station.

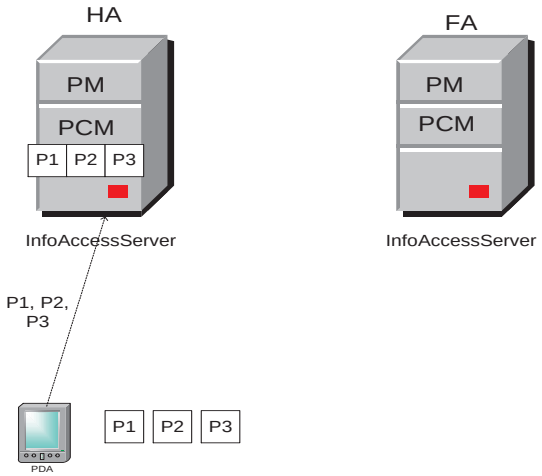
3.3.3 Interaction between Profile Manager and Profile Cache Manager

Interaction between profile manager and profile cache manager of a mobile client can be well explained with the help of the Figure 3.5.

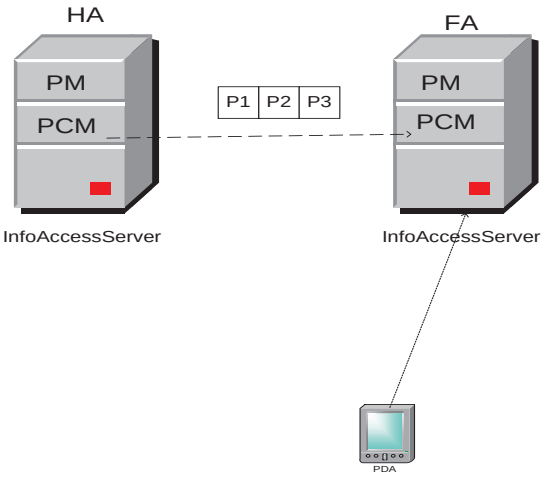
Figure 3.5(a) shows a mobile client that is connected to its home agent. The mobile client submits three profiles P1,P2 and P3 to the profile manager. Let us assume that all profiles represent long-term periodic queries and they need to start executing immediately. The profile manager validates the profiles and sends it to the profile cache manager on the HA itself. The profile cache manager starts executing the profiles.

Figure 3.5(b) shows a mobile client that has just moved to a foreign agent. The profile cache manager on the HA decides whether or not to move the profile cache. Let us assume that it decides to move the profile cache. It sends the profile cache to the profile cache manager on the FA, which will then continue executing the profiles.

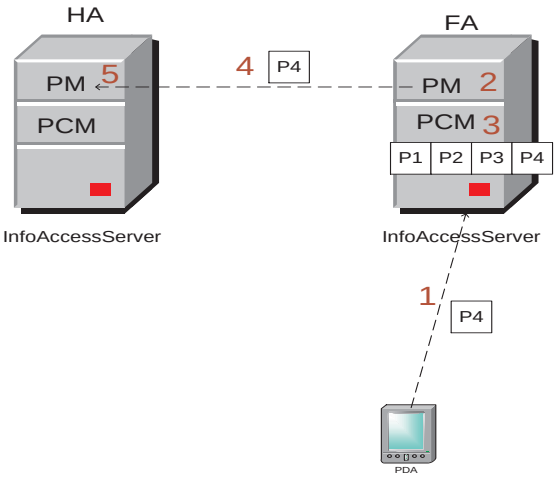
Figure 3.5(c) shows a mobile client that is serviced by a foreign agent. The mobile client wants to submit a new profile that has to start executing immediately. For simplicity, let us assume that the profile cache of the mobile client is currently executing in the profile cache manager on the same FA. The following steps take place:



(a) Submitting profiles when the user is serviced by HA



(b) Migration of profile cache to a FA



(c) Submitting profiles when the user is serviced by FA

Figure 3.5 Interaction between Profile Manager and Profile Cache Manager

1. The mobile client submits the new profile to the profile manager on the FA as shown in step 1 on the Figure 3.5(c).
2. The profile manager checks to see if the profile is scheduled immediately and also to check if the profile cache for the mobile node is executing on the profile cache manager that is deployed on the same FA. Based on the above conditions, there can be the following two cases:
   (a) If one of the conditions is false, the profile manager of the FA sends the new profile to the profile manager of the HA and stops doing any execution with regard to the new profile. In this case, the profile manager on HA takes care of starting the execution of new profile. At the elapse of start time, the profile manager on HA sends the profile to the profile cache manager that is executing the profile cache for the mobile node at that time.
   (b) If both the conditions are true, the profile manager validates the new profile and sends it to the profile cache manager on the same FA as shown in step 2 of the Figure 3.5(c).
3. The profile cache manager starts executing the profile as shown in step 3 of the Figure 3.5(c).
4. The profile manager on the FA sends the profile to the profile manager on HA with a flag indicating that the profile has started execution as shown in step 4 of the Figure 3.5(c)
5. The profile manager on HA stores the information regarding profiles in its profile table as shown in step 5 of the Figure 3.5(c).

3.4 Summary

InfoAccess is a distributed proxy-based scheme that offers efficient information retrieval for mobile clients. InfoAccess servers accept the profiles, process the results

according to the specification of profiles and notify results to users in a timely manner. In InfoAccess, profile cache is dynamically migrated from one InfoAccess server to another to maintain the execution of profiles close to the user.

CHAPTER 4

PROFILE PROCESSING AND GROUPING

In InfoAccess, profiles have to be processed. To process profiles, data has to be fetched from the data sources, personalized results have to be computed from the fetched data and the results have to be notified to the user. All these activities have to be performed sequentially and can be considered as events that trigger the next event. Therefore, profiles are represented as event graphs called the query graphs. Query graphs provide an efficient mechanism to represent and process profiles. This representation also facilitates grouping of similar queries. This chapter discusses the construction of query graphs and the processing and grouping of profiles using query graphs.

4.1 ECA paradigm

Every valid profile that is submitted by the user initiates a series of operations that have to occur at different points in time. Some of these operations are starting the execution of the profile, fetching the data from the data sources, computing the results, notifying the user and stopping the execution of the profile. ECA rules [11, 14] provide an elegant mechanism for supporting execution of these operations based on events. InfoAccess servers generate ECA rules to perform these operations.

Briefly, an event-condition-action rule has three components: an event (occurrence of an event), a condition (checked when the associated event occurs), and an action (operations to be carried out when the condition evaluates to true). The ECA paradigm has been used for monitoring the database state in active databases and as a stand-alone concept for monitoring objects in applications (both centralized and distributed

[15]). As part of the active object-oriented system, a local event detector (LED) has been developed as a library that can be used to declare events and associate rules to be executed when events occur in a seamless manner [12, 16]. It is actually an event detector that has been implemented to detect events in java applications and execute rules defined on them. Primitive events (as method executions) and temporal events (both absolute and relative time), as well as composite events are supported in LED. InfoAccess uses periodic event operator to support periodic result computation for periodic queries and PLUS operator for activation and deactivation of profiles. This section provides an overview of the tools/components that are used in InfoAccess and discusses how ECA rules are used for: i) activation and deactivation of profiles, and ii) generating rules for computing results.

4.1.1 Activation and Deactivation of Profiles

In InfoAcess, a profile has a lifespan bounded with a start time and an end time. During the profile's lifespan, a profile is active and the results are computed for the profile. A profile is activated at its start time, and can be deactivated explicitly by the user during its lifespan or the profile is deactivated at the end time of the lifespan. The start/end time of the profile is a time point on the time line. When a profile's lifetime is short, medium or long, it is activated immediately. But in cases where the start is at a later time point, activation of the profile is deferred until that time elapses. To facilitate this, we need a triggering mechanism that will raise events required to activate/deactivate a profile. In InfoAccess, the scheduler module of the profile manager generates appropriate events and rules and are instantiated using the LED. The start and end events are implemented as primitive events. For every profile, start and end events are created and rules are associated.

Event $Start\_p_i$ = createEvent("$start\_p_i$") (1)

Rule $Rstart\_p_i$ = createRule($Start\_p_i$, $condition\_p_i$, $action\_p_i$) (2)

Statement 1 shows the start event creation and statement 2 shows the rule creation for a profile p_i. When the event created in statement 1 is raised, the rule associated with it (statement 2) is triggered. When the rule is triggered the action is performed only when the condition is true. With activation/deactivation of profiles there are no conditions to check, thus when the start event is raised, the respective rule activates the profile. By activation, we mean that the profile manager sends the profile to the profile cache manager that currently executes the profile cache for the mobile client. An event can be raised in the following ways.

- Absolute Time : Consider the scenario where p1 is defined in the interval [11/04/04, 11/05/04]. At time 11/04/04 the start event associated with profile p_1 has to be raised for it to get activated. Following are the events and rules that are generated to activate profile p1:

  Event $Start\_p_1$ = createEvent("$start\_p_1$")

  Rule $Rstart\_p_1$ = createRule($Start\_p_1$, $condition\_p_1$, $action\_p_1$)

  Event $ETime_1$ = createTemporalEvent (11/04/04)

  Rule $RTime_1$ = createRule ($Etime_1$, condition, action)

  When event $ETime_1$ is raised at the specified time point, rule $RTime_1$ is triggered. The action associated with rule $Rtime_1$ in turn raises the event $Start\_p_1$. When $Start\_p_1$ is invoked, the action ($action\_p_1$) associated with rule ($Rstart\_p_1$) activates profile p_1.

- Relative : The end time of a profile may depend on the start time of the profile. This happens when the lifespan of the profile is specified as short, medium or long. End time of the profile which has a short lifespan is a day after the start time. Similarly, for medium lifespan profiles, end time is a week after the start time and

for long lifetime profiles, end time is a month after the start time. The following are the events and rules that are generated in order to activate and deactivate a profile p1 with short lifespan.

Event $Start\_p_1$ = createEvent("$start\_p_1$")

Rule $Rstart\_p_1$ = createRule($Start\_p_1$, $condition\_p_1$, $action\_p_1$)

Event $ETime_1$ = createTemporalEvent (11/04/04)

Rule $RTime_1$ = createRule ($ETime_1$, condition, action)

Event End_p_1 = createEvent("end_p_1")

Rule $Rend\_p_1$ = createRule(end_p_1, $condition\_p_1$, $action\_p_1$)

p1Plus_1day = createPlusEvent ($Start\_p_1$, 1 day)

Rule $R\_End\_p_1$ = createRule (p1Plus_1day, condition, action)

The event End_p1 is raised by the rule $R\_End\_p_1$ when the plus event p1Plus_1day is raised. And finally profile p_1 is deactivated. Each profile generates a start and an end event. Rules are associated with these events to trigger activation/deactivation of the profiles.

4.1.2 Generating Rules for Computing Results

In InfoAccess, the user can specify a one-time query or a periodic query using the notify option. A one-time query has no lifespan and will just have one event to activate the profile. The rule associated with this event fetches the data from the data sources, computes the result and notifies the user. After this, the profile is deactivated and no further processing is done for this profile. However, when the user specifies a periodic query, the data has to be fetched at the specified periodicity to compute the results. In InfoAccess, the periodic event of LED is used to achieve this in an asynchronous manner. A periodic event is an event that repeats itself within a constant and finite amount of

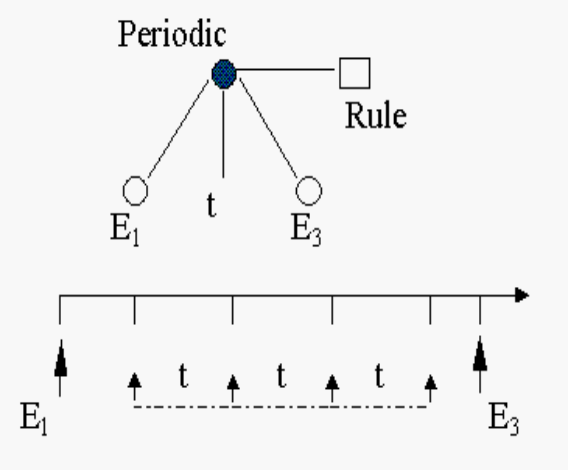


Figure 4.1 Periodic Event

time. Figure 4.1 shows the graphical interpretation of the periodic event of LED. The initiator and terminator are the start (E_1) and end (E_3) events of a profile and t is the interval with which the results are to be computed. The actual fetch of data from the data sources and the computation of results are performed by the rule associated with the periodic event. Suppose the user wants to know temperature every hour, a periodic event with an associated rule to fetch weather webpage and compute temperature information is generated. A periodic event and the associated rule for the above scenario is as follows.

Event $FetchEvent\_p_1$ = createPeriodicEvent ($Start\_p_1$, 1 hour, End_p_1)

Rule $FetchRule\_p_1$ = createRule ($FetchEvent\_p_1$, condition, action)

In this manner, ECA rules are used to asynchronously activate and deactivate profiles at run time. Once the appropriate events and rules are created, the local event detector (LED) handles the execution at run time. By activation/deactivation of profile, we mean addition/deletion of that profile to the query graphs that is detailed in the next section. The query graph modifier module appropriately generates periodic events and associated rules to fetch the data, compute results and notify the user.

4.2 Query Graphs

For every profile, data needs to be fetched from the data sources, then the data has to be preprocessed to compute the results required by the user and then delivered to the user. For simple queries, preprocessing would be straight forward. However for complex queries, preprocessing the data becomes complex as the system has to keep track of various information that is required by the user. If we consider fetching the data, preprocessing and notification as events, an event graph in the form of query graph can be maintained for each profile to perform efficient processing and notification. In the query graph, fetching of data is considered an event that triggers preprocessing and preprocessing is an event that triggers notification.

Query graphs should be built such that the following conditions are satisfied:

- Query graphs should allow us to asynchronously feed fetched data for further processing.
- They should allow parallelism where possible.
- They should optimize computation by grouping profiles over data sources.
- They should facilitate composite result computation using the same paradigm that is used for primitive result computation.

Primitive result computation is the computation involved in accessing specific information. For example, computing the temperature from a weather webpage is a primitive result computation. Composite result computation is the computation that involves more than one primitive result computation related through one or more event operators such as NOT, AND, OR. For example, the user might require "Temperature AND Humidity" from the weather webpage. Also, deletion and propagation of delete semantics must be straightforward in the representation chosen.

For every profile, query graph is generated by the "query graph modifier" module. The query graph is constructed bottom-up. The leaf node represents the data source and

is called the "datasource node". At level-1 of the query graph are the "filter nodes", which filter out the information of interest to the user. Composite nodes represent composite result computation and form the nodes at level-2 and above.

Data flows through the event graph in a bottom-up fashion. Typically, in a query graph higher-level nodes subscribe to lower level nodes in the graph. This subscription information is maintained in the subscriber list at each node. Profiles that need to be notified are also maintained in the subscription list. When information is available at a node, it notifies all its subscribers. Therefore, the information is either propagated upwards to a node for further processing or the information is notified to the user.

Although based on the same concept of event graphs explained above, the construction of a query graph for a database source is a little different from the one for a webpage source because of the fundamental difference in the way each of these data sources make the data available. The webpage provides information on a single page and the system has to scan the page for the required information, whereas in a database, tuples that satisfy certain filter conditions are made available. Therefore, for a webpage data source, the system has to fetch data in the webpage based on the keywords specified in the keyword clause and then apply filter conditions. However, for a database data source, the system has to fetch data (tuples) based on the filter conditions and then extract keywords (project attributes).

4.2.1 Building a query graph for a webpage data source

In a query graph for a webpage, the data source node denotes the URL from which the information needs to be fetched. At level-1 are the filter nodes that represent the keywords in the keyword clause of the profile. The filter condition is also incorporated in the filter node. The filter condition clause contains one or more filter conditions connected by AND, OR and NOT. Composite nodes are created to facilitate composite

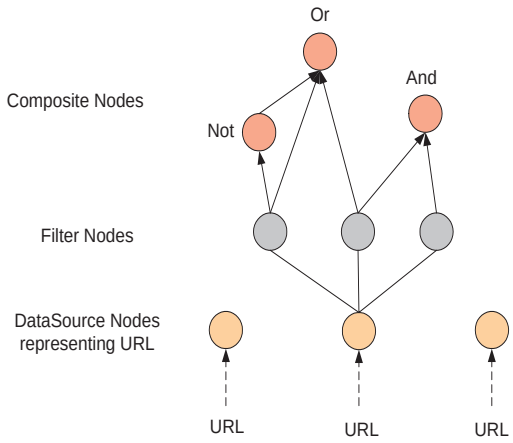


Figure 4.2 Generalized query graph for a webpage

result computation. When the data source is a webpage, the semantics of the composite nodes are similar to the semantics of the composite events in LED [17]. The adapted semantics of AND, OR and NOT are as shown in the table 4.1.

Table 4.1 Composite Nodes for a Webpage

Composite Node	Semantics
NOT	Filter condition not satisfied ($\neg\ C_1$), where C_1 is the filter condition
OR	Either of the filter conditions are satisfied ($C_1 \vee C_2$), where C_1 and C_2 are filter conditions
AND	Both the filter conditions are satisfied ($C_1 \wedge C_2$), where C_1 and C_2 are filter conditions

Figure 4.2 shows a generalized query graph for a webpage data source. In general, the keywords without any filter conditions imposed on them will be connected with "AND" nodes as all the information represented by the keywords are required. However, when there are filter conditions that themselves are connected by AND, OR or NOT,

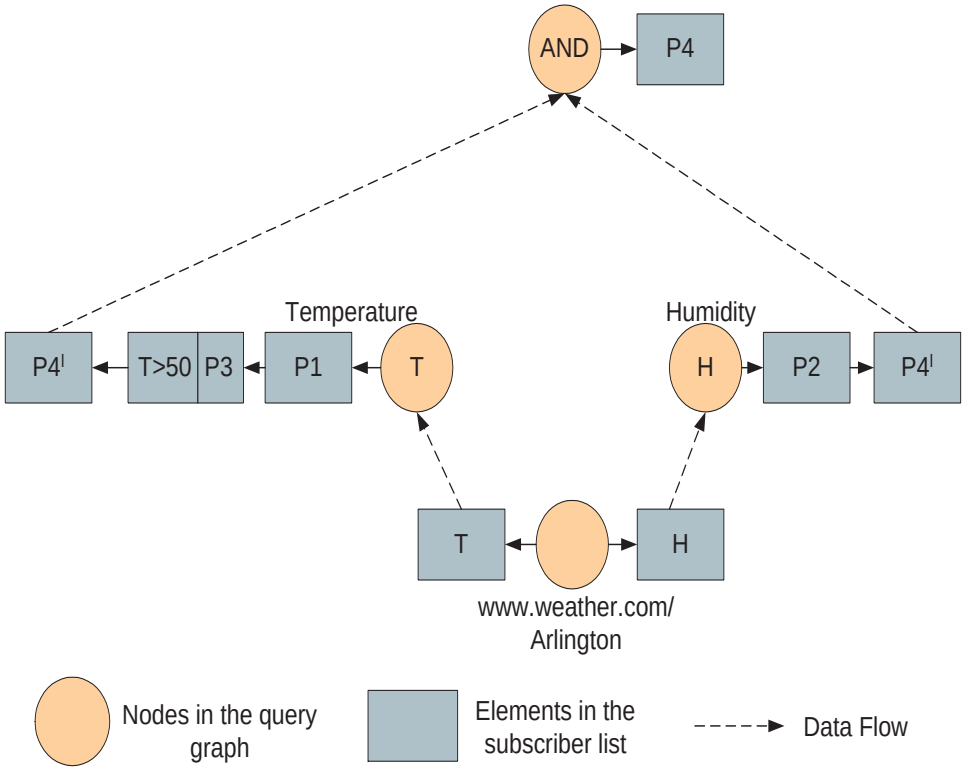


Figure 4.3 Example illustrating query graph for a webpage

the same composite nodes will be imposed on the keywords that are represented by filter nodes.

4.2.1.1 Illustration of constructing a query graph for a webpage

Consider the following profiles that need information from the www.weather.com webpage for the same location. For simplicity, let us assume that they have the same lifespan and same periodicities.

P1 - Temperature

P2 - Humidity

P3 Temperature , if temperature > 50

P4 Temperature, Humidity

Figure 4.3 shows the query graph that is constructed for the profiles. The weather webpage along with the location forms the URL that would be represented in the datasource node. For instance if the location is Arlington, TX, the datasource node would represent the URL www.weather.com/Arlington. The keyword specified by P1 is Temperature. Therefore, a filter node with "Temperature" would be created at level-1 and the profile P1 is added to the subscriber list of the "Temperature" node so that whenever temperature is available, it is notified to profile P1. Similarly, "Humidity" filter node is created for profile P2. Even though P3 requires temperature, there will be no new "Temperature" node created for profile P3 as the existing one can serve the purpose. The filter condition, "Temperature $>$ 50" is checked before delivering the results to the user. This is done to avoid the recomputation of temperature every time temperature is required with a new filter condition. Profile P4 requires temperature and humidity. Therefore, the "Temperature" and the "Humidity" nodes are joined by a composite "AND" node. The profile P4 is put in the subscriber list of "AND" node so that the user is notified when both "Temperature" and the "Humidity" are available. This example illustrates the benefit of sharing common computation besides showing the construction of the query graph. We can see that the page is fetched only once for all the profiles that require the same webpage instead of fetching it once for every profile. The computation of information at level-1, level-2 and so on are also shared in the same way.

4.2.2 Building a query graph for a database data source

In a query graph for a database, the datasource node represents the table in the relational database. The filter nodes represent the predicates specified in the filter conditions clause of the profile. Level-2 and above nodes consist of composite nodes. Figure 4.4 shows the generalized query graph for a database.

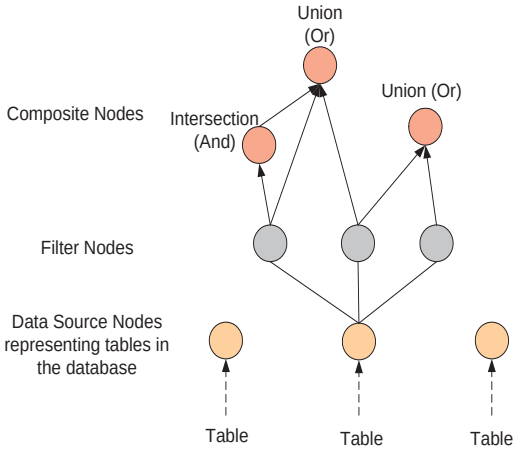


Figure 4.4 Generalized query graph for a database

The composite nodes in the context of database are different from the ones that we explained for the webpage datasource (AND, OR and NOT). Even though the filter condition expression is specified by the user using AND, OR and NOT, the semantics are different. In databases, if the two filter conditions are connected by "OR", it actually represents the union of tuples that satisfy both the conditions, whereas in the webpage data source, "OR" tells the system that the information has to be notified if one of the filter conditions is satisfied. Thus, we have a new composite node called "UNION" to represent "OR". If the two filter conditions are connected by "AND", it means that the tuples obtained from the first condition can be filtered to obtain the tuples that satisfy the second condition. Here, there is no need to make a database call for obtaining the tuples that satisfy the second condition as the set of resultant tuples is a subset of the tuples obtained from the first condition. Therefore, we have a new composite node called "INTERSECTION" to represent "AND". The semantics of "UNION" and "INTERSECTION" are described in the table 4.2. There is no equivalent of "NOT" composite node, as "NOT" will be specified with the filter condition in the filter node while extracting the tuples. The tuples at the final node are filtered for the project

attributes that are specified in the keyword clause of the profile, before delivering the results to the user.

Table 4.2 Composite Nodes for a Database

Composite Node	Semantics
UNION	Union of the set of tuples that satisfy the filter conditions (T1 ∪ T2), where T1 and T2 are the set of tuples that satisfy the filter conditions C1 and C2
INTERSECTION	Intersection of the set of tuples that satisfy the filter conditions (T1 ∩ T2), where T1 and T2 are the set of tuples that satisfy the filter conditions C1 and C2

4.2.2.1 Illustration of constructing a query graph for a database

Consider the following profiles that need information from the restaurants database. For simplicity, let us assume that they have the same lifespan and same periodicities. The restaurants database has a table called "restaurant". The restaurant table has the attributes ID, restaurantName, cuisine and location. Let the profiles represent the following sql queries:

P1 → Select * from restaurant where location = "Arlington,TX"

P2 → Select * from restaurant where location = "Arlington,TX" or location = "Irving,TX"

P3 → Select * from restaurant where location = "Arlington,TX" and cuisine = "American"

The query graph for the above profiles is as shown in the Figure 4.5. The table "restaurant" forms the datasource node. A filter node for the predicate "location = Arlington,TX" is added for P1 and P1 is maintained in the subscriber list of this node. For P2, only a filter node for "location = Irving,TX" is added, as the filter node "loca-

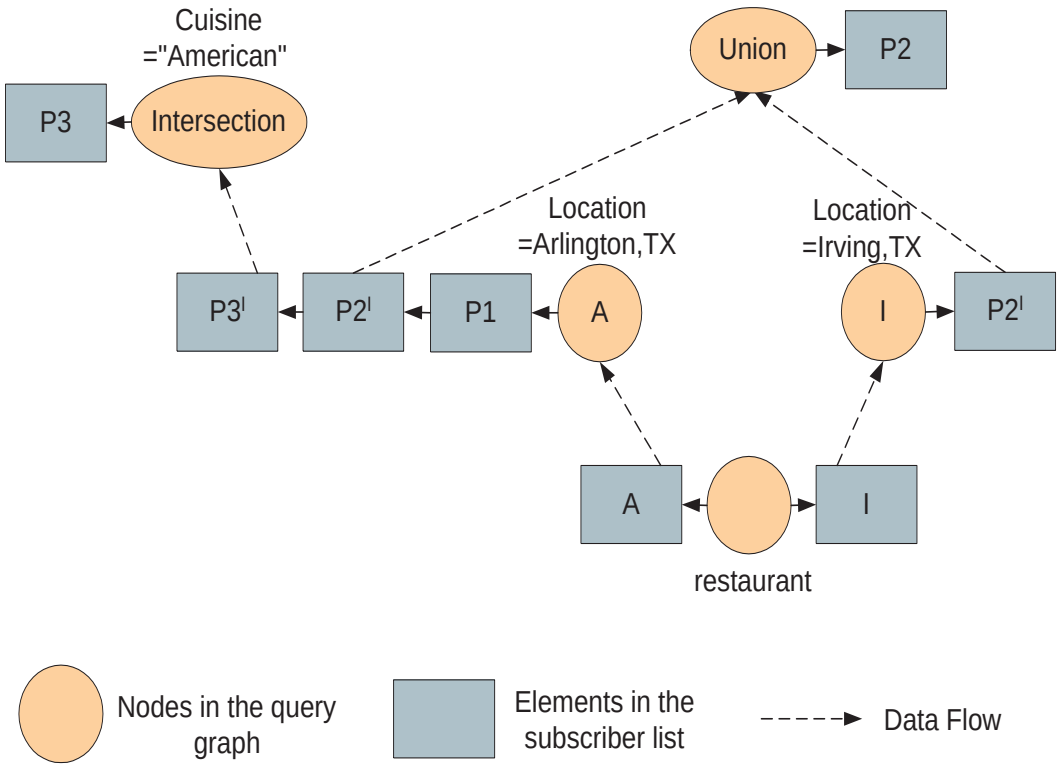


Figure 4.5 Example illustrating query graph for a database

tion = Arlington,TX" can be reused. As P2 requires the union of the tuples obtained from the predicates "location = Arlington,TX" and "location = Irving,TX", they are connected by an "UNION" node. P2 is added to the subscriber list of "UNION". For P3, tuples obtained at the filter node "location = Arlington,TX" need to satisfy another condition "cuisine = American". Therefore, an "INTERSECTION" node representing the predicate "cuisine = American" is added to the query graph and P3 is added to the "INTERSECTION" node as a subscriber. This example illustrates the building of a query graph and sharing of computation when the data source is a database.

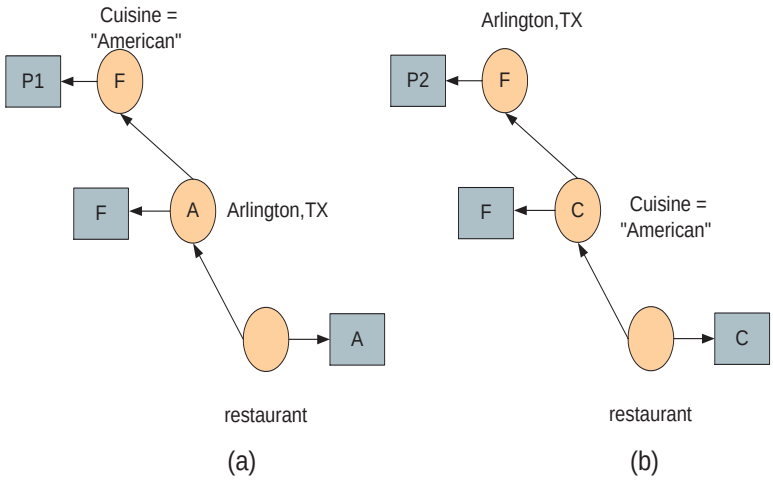


Figure 4.6 Handling "AND"

4.2.2.2 Handling "AND"

Multiple "AND" nodes in the filter conditions clause give rise to single path trees in the query graphs for databases. This could lead to the construction of two different graphs that compute the same result. Therefore, an order has to be maintained on filter conditions when constructing a query graph. The problem is well explained using an example. Suppose a user wants to know restaurants which satisfy conditions *"location = Arlington,TX AND cuisine = American"*. The query graph shown in the Figure 4.6(a) is constructed. If another user wants to know restaurants that satisfy the conditions *"cuisine = American AND location = Arlington,TX"*, another query graph is constructed to compute the same result as shown in the Figure 4.6(b). This can be avoided if we maintain an order on filter conditions. These filter conditions are referred to as predicates in the database world. We assume a predicate order based on the selectivity of the predicate. The selectivity of the predicate can be obtained by querying the database schema. The predicates that are connected with "AND" are first sorted in an ascending order and then the query graph is built. So, predicates with lower selectivity factor lie

below the predicates with higher selectivity factor in the query graph, thereby reducing the number of tuples at each node.

4.3 Grouping similar Profiles

At a given instance of time, an InfoAccess server will be processing profiles for a large number of mobile clients. This number will be close to the number of mobile clients that are serviced by the BS on which the InfoAccess server has been deployed. We assume that each of the mobile users have considerable number of location-dependent profiles that are submitted to the server. As the profile cache for the mobile client is made location-aware by migrating the profile cache dynamically to follow the movement of the user and the mobile clients are querying for local information, there will be large number of similar profiles. Similar profiles must be grouped together so that redundant computation can be avoided. Consider a scenario where there are n users querying for local weather information. Instead of doing the profile processing n times, it can be done only once by grouping them together. Grouping not only reduces the number of times data is fetched from the data sources but also reduces the high computation involved in preprocessing the data to customize it to the needs of the user. Profile grouping is an optimization technique developed to minimize computation. Grouping of profiles is based on data sources, information they are interested in and periodicity.

Profiles are grouped when there is more than one profile that require data from the same data source. The way the query graphs are constructed allows us to determine if the new profile is querying for the same information. Examples shown in Figures 4.3 and 4.5 clearly show how the query graphs have been used to share the computation done by a node. The only complexity that was not handled there was the one that arises due to the differences in periodicities. Periodicities for two profiles might be different but might still fall on coinciding time points on the time line. For instance, profiles P1, P2

and P3 with t, $2t$ and $3t$ periodicities fall on the same time points on the time line. Each such group will have only one periodic event with periodicity t. Although all the three profiles need the same information, P1 needs it every t time period, P2 needs it once in two t time periods and P3 needs it once in three t time periods. This information has to be maintained in order to notify results at the right interval to various users. Therefore, there is a need for a data structure that can keep track of the group and information regarding group members.

The grouping data structure keeps track of the periodicity of the group, all the profiles in the group and a count associated with each profile. The count gives the number of time periods after which the information has to be sent to the user. Initially, count is set to a number equal to *(Periodicity of the profile/Periodicity of the group)*. The count is decremented by one every time period. When the count reaches zero, information is sent to the user and the count is reset to the initial value. The grouping data structure also maintains a table that is indexed by the keyword (webpage) or the predicate (database) in the filter node. This table maintains all the profiles that need information on a particular filter node along with the count. This can be better explained with an illustration that follows.

4.3.1 Illustration showing the data flow in query graphs and grouping data structure

Consider the following profiles that have been grouped together.

P1 - Temperature in Arlington, TX every t time period

P2 Temperature in Arlington, TX every $2t$ time period

The query graph, the grouping data structure and the periodic event for the group are as shown in the Figure 4.7. A periodic event with time t is generated for the group. The grouping structure maintains temperature in the table which points to a linked list

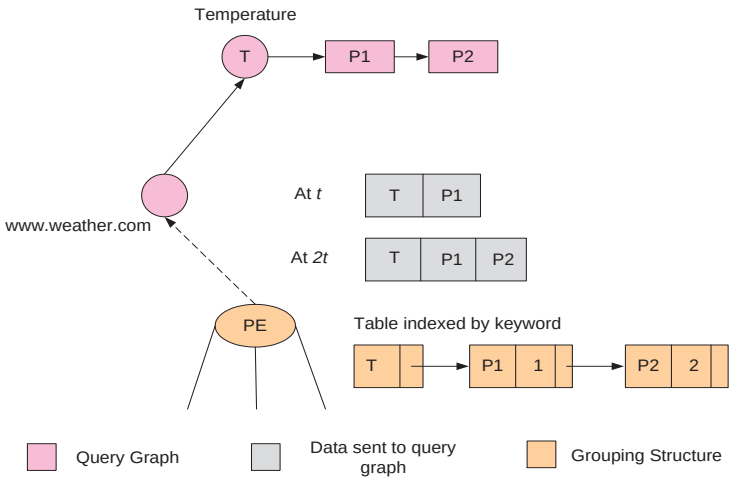


Figure 4.7 Example illustrating grouping data structure

containing the profile P1 and P2 with counts 1 and 2 associated with them. After the elapse of one t time period i.e., after the periodic rule is fired for the first time, the temperature is encapsulated with the profile P1 and sent to the datasource node. The datasource node then decapsulates the temperature, P1 and sends P1 to the "Temperature" filter node. The "Temperature" node then notifies the user of P1. Similarly, after two time t periods, the temperature is encapsulated with both the profiles P1 and P2 and eventually both P1 and P2 are notified. Thus, the query graph, the periodic event and the grouping data structure work in harmony to provide the information needed by the user at the right time.

4.3.2 Issues with grouping

Profiles representing periodic queries have periodicities specified in them. All profiles that have the same data source and have periodicities that fall on coinciding time points on the time line share the periodic event and the rule associated with it. Although profiles can be grouped as stated above, there are other issues that should be addressed. The issues are explained below in detail.

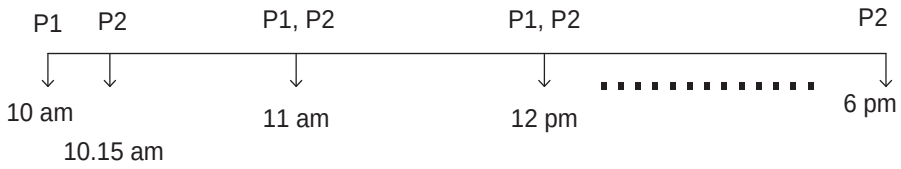


Figure 4.8 Notification times for P1 and P2

1. The start times and end times of the profiles need not be the same even when the periodicities fall on coinciding time points on the time line. Suppose there are two profiles requiring weather information every hour with the following lifespan:

   P1 with interval [10am, 5pm]

   P2 with interval [10.15am, 6pm]

   When P1 is activated, a periodic event that has a start time event at 10am and an end time event at 5pm with 1 hour periodicity is created. When P2 comes in, because the start time is 10.15am, it requires information at 10.15am, 11.15am and so on, which does not exactly coincide with the time points of P1 (10am, 11am and so on). To increase the number of profiles that can be grouped, we ignore the start time of P2 and group P2 along with P1 as the periodicities (without taking start times into consideration) fall on coinciding time points. But, we ensure that the user of P2 gets the required information every hour. To do this, as soon as P2 starts executing at 10.15am, results for P2 are computed and sent to the user. After that, the results for both P1 and P2 are computed at 11am, 12am and so on. The notification times for P1 and P2 are as shown in the Figure 4.8.

   There is another problem associated with the end events. In the above example, P1 ends first at 5pm along with which the periodic event ends. But, results need to be computed for profile P2 even after P1 ends. When P1 ends, the end time event has to be replaced appropriately.

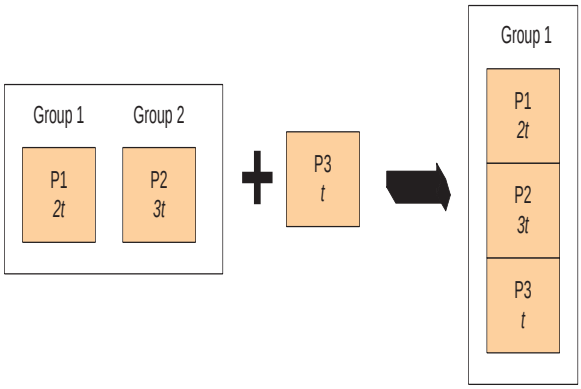


Figure 4.9 Merging two groups

When there are more than two profiles in the system that belong to the same group, the periodic event initiators(start time events) and terminators(end time events) have to be determined at runtime and if needed, be replaced at runtime. In order to avoid this computation each time a profile belonging to the same group is registered, dummy initiator and dummy terminator are created. The periodic event generated would be

$FetchEvent\_Page_i$ = createPeriodicEvent (Start_dummy, t, End_dummy),

where t is the minimum of the periodicities of all the profiles.

When the start of a profile is raised the rule associated with it checks on the status of the periodic event (i.e., initialized). If initialized it does not raise the Start_dummy event. Similarly when the end of the sentinel is raised it checks whether the Fetch rule is servicing other profiles. Based on this information the End_dummy event is raised.

2. Another issue that needs to be addressed is the one with periodicities. If a profile P1 with periodicity *2t* is activated first and another profile P2 with periodicity t gets activated after P1, both P1 and P2 belong to the same group but, the periodic event generated when P1 was activated has a periodicity of *2t* even though P2

requires results t time period. Therefore, when P2 gets activated, the periodicity of the periodic event is changed. The general rule is to change the periodicity of the group when a new profile with a periodicity lower than the periodicity of the group joins the group.

3. Finally, there is the problem of "merging and splitting" that is explained below. Suppose there are two profiles P1 and P2 with periodicities *2t* and *3t*. They cannot be grouped together and so they will be placed in two different groups. But, when a third profile P3 with periodicity t enters the system, P1, P2 and P3 can be grouped. This is handled by merging the two groups, thus creating only one group with periodicity *t*. This is illustrated in the Figure 4.9. Although this computation is expensive, it occurs very infrequently. We need to check if any of the existing groups can be merged only when a profile with a periodicity lower than the lowest of the periodicities of all the groups enters the system. Similarly, when a profile with the lowest periodicity in the group leaves, the group has to be checked to see if the group needs to be split. In the above example, if the profile with periodicity t leaves the group, the group has to be split into two groups of periodicities *2t* and *3t*.

4.4 Deletion of Profiles

When the profile is moved to another BS or when the profile is disabled or when the lifetime of a profile ends, the profile has to be deleted from the query graphs, the grouping data structure and the tables. Deleting the profiles from the tables is straight forward. However, deleting them from the grouping data structure and the query graphs need some discussion. Deleting the profile from the grouping data structure involves deleting the entries for the profile in the grouping data structure and checking to see if the group needs to be split.

Deletion of the profile from the query graph is done in a top-down fashion. The profile is deleted from the subscriber's list at the highest level node. If there are no more subscribers in the subscriber's list of the node, the node is deleted. Then the node at next lowest level is considered and the same operations are done to the node. This is continued until we find a node with a subscriber or all the nodes in the query graph are deleted.

4.5 Summary

Query graphs are used to efficiently process profiles and notify results to the user. Query graphs also facilitate grouping of similar profiles so that common computation can be grouped. Active capability (ECA rules) is used to support the processing of scheduled and periodic queries.

CHAPTER 5

MIGRATION OF PROFILE CACHE

In InfoAccess, active profiles of a mobile client are managed and maintained by the profile cache manager as a profile cache. The profiles are executed on the same profile cache manager where the profile cache is maintained. The challenge is to find the right profile cache manager on the network to maintain and execute the profiles in the profile cache. Profile cache can be maintained on the profile cache manager of the home agent or on a profile cache manager on the foreign agent currently servicing the mobile client. This chapter discusses the pros and cons of the above two approaches and proposes a hybrid approach that derives the best of both the approaches.

5.1 Profile Cache on HA

In this approach, the profile cache is stationary at the profile cache manager of the home agent. As the user moves away from the home agent, the result notification costs increase. Imagine a user who submits a long-term periodic query in the form of profile on his/her home agent in Los Angeles (LA) and then moves to New York (NY). In this case, the result has to be notified to the user in NY consuming a lot of bandwidth across the network. But, the obvious advantage of this approach is that no result notifications are lost due to profile cache movement as the profile cache is never moved from the home agent.

5.2 Profile Cache on FA

In this approach, the profile cache is moved to the FA that is currently servicing the user. Therefore, the profile cache has to be moved every time the user moves to a new BS. Here, the result notification cost is reduced considerably. Also, it has a unique advantage because the profile cache is in the same location as the user, making the profile-cache location aware. There will be considerable number of users looking for local information and so will submit similar profiles to the same profile cache manager. For example, many users would want to know the weather information in their locality. The mere fact of having a number of similar profiles that need to be processed on the same profile cache manager can be exploited to optimize computation. Similar profiles can be grouped so that they share common computation.

On the other hand, moving the profile cache every time the user moves to a new BS can be an overkill. A fast moving user changes BSs frequently, resulting in significant movement of the profile cache. In such cases, the profile cache is in movement for most of the time and so is unavailable to the user. Another problem with moving the profile cache every time the user changes a base station is the ping-pong movement of the profile cache. In this case, the profile cache is continuously transferred between the two base stations, essentially making the profile cache unavailable to the user.

5.3 Hybrid Approach

The above mentioned methods are extremes and therefore there is a need for an hybrid approach where the best of both the approaches are adapted. The profile cache can be moved to a new BS only if it reduces the data transmission cost across the wired network instead of moving it every time the user changes a BS. This would keep the profile cache close to the user making it location-aware. It would also reduce the result

notification costs. At the same time, the profile cache is available to the user most of the time as it is in movement for lesser amount of time. As the profile-cache is location-aware, similar profiles need to be computed on the same profile cache manager. So, grouping can be done to optimize computation.

Every time the user moves to a new BS, decision has to be made as to whether the profile cache should be moved. Various parameters that affect migration of profile cache are analyzed. The profile cache is moved to the new BS only if it reduces the bandwidth usage across the wired network. The analysis is explained in detail below. First, the cost factor for a single profile is derived. Based on this, the total cost factor for all the profiles in the profile cache is found. The total cost factor is used by the "decision maker" module to decide if the profile cache should be moved to the new BS.

5.3.1 Cost Factor for a Single Profile

The cost factor for a single profile is found by comparing the cost of moving the profile versus the cost of not moving the profile.

Let,

$$C_{Moving} = Cost\ of\ moving\ the\ profile\ cache$$

$$C_{NotMoving} = Cost\ of\ not\ moving\ the\ profile\ cache$$

Cost is measured in units of time. Intuitively, the profile cache should be moved only if $C_{Moving} < C_{NotMoving}$. Therefore, we find C_{Moving} and $C_{NotMoving}$.

$$\begin{aligned} C_{Moving} \;=\; & Cost\ of\ deleting\ the\ profile\ cache\ in\ the\ old\ base\ station\ (C_{del}) \\ & + Cost\ of\ adding\ the\ profile\ cache\ in\ the\ new\ base\ station\ (C_{add}) \\ & + Cost\ of\ transferring\ the\ profile\ cache\ from\ old\ base\ station\ to\ the \\ & \quad new\ base\ station\ across\ the\ wired\ network\ (C_{transfer}) \end{aligned}$$

i.e., $C_{Moving} = C_{del} + C_{add} + C_{transfer}$

C_{del} involves the cost of stopping the execution of the profiles belonging to the profile cache and deleting them from the data structures (query graphs) that help in the processing of profiles. Similarly, C_{add} involves building the data structures for processing of profiles and starting the execution of the profiles. As C_{del} and C_{add} are CPU computations, they are in the order of microseconds whereas transferring data over the wired network is in order of milliseconds. Therefore,

$C_{del} + C_{add} << C_{transfer}$. Hence,

$$C_{Moving} = C_{transfer} \tag{5.1}$$

As the profile along with the cached result has to be moved across the network,

$$C_{transfer} = (Size\ of\ the\ profile\ in\ KB\ (S_{profile}) + Size\ of\ the\ result\ in\ KB\ (S_{result})) * Cost\ of\ transferring\ a\ KB\ of\ data\ across\ the\ network\ (C_{KB})$$

Substituting for $C_{transfer}$ in equation 5.1, we have

$$C_{Moving} = (S_{profile} + S_{result}) * C_{KB} \tag{5.2}$$

If the profile has not moved, it means that the old base station has to push results to the new base station to deliver results to the mobile client. Therefore,

$$C_{NotMoving} = Estimated\ number\ of\ data\ pushes\ (N_{est}) * Cost\ of\ transferring\ the\ result\ (C_{result})$$

$$C_{result} = S_{result} * C_{KB}$$

$$C_{NotMoving} = N_{Est} * S_{result} * C_{KB} \tag{5.3}$$

From equations 5.2 and 5.3,

$Cost\ Factor\ =\ C_{NotMoving}\ /\ C_{Moving}$

$Cost\ Factor = ((S_{profile} + S_{result}) * C_{KB})/(N_{est} * S_{result} * C_{KB})$

$$Cost\ Factor = (S_{profile} + S_{result})\ /\ (N_{est} * S_{result}) \tag{5.4}$$

By observing equation 5.4, the only parameter that changes is N_{est} and the other parameters are constant for a given profile. Also, when Cost Factor = 1, $C_{NotMoving} = C_{Moving}$. Therefore, when Cost Factor = 1, we can find a cutoff point (N) for the number of data pushes after which moving the profile becomes beneficial.

$1 = (S_{profile} + S_{result})\ /\ (N * S_{result})$

$N = (S_{profile} + S_{result})\ /\ S_{result}$

If $N_{Est} > N$, the profile should be moved to the new BS. Therefore,

$$Cost\ Factor = N_{Est}\ -\ N \tag{5.5}$$

If the *Cost Factor* is positive, the profile should be moved.

Estimated number of data pushes (N_{Est}) can be calculated by predicting the amount of time that the user will stay in the new base station (based on the history) and the time period after which the results are to be delivered.

$N_{est}\ =\ Estimate\ of\ time\ the\ Mobile\ client\ will\ stay\ in\ new\ BS\ /\ time\ period\ of\ the\ profile$

Estimate of time that the mobile client will stay in new BS can be found out by collecting the history of the time the mobile client has stayed in previous base stations. We predict this by using the exponentially weighted moving average so that appropriate leverage can be given to the history as well as the current data that is available.Let,

$T_{avg}\ =\ Average\ time\ that\ the\ mobile\ client\ has\ stayed\ in\ a\ BS$

$T_{cur}\ =\ Time\ that\ the\ mobile\ client\ has\ stayed\ in\ the\ current\ BS$

$T_{new}\ =\ Estimate\ of\ the\ time\ that\ the\ mobile\ client\ will\ stay\ in\ the\ new\ BS$

Using exponentially weighted moving average,

$$T_{new} = \alpha * T_{cur} + (\alpha - 1) * T_{avg} \text{ , where } \alpha \text{ ranges between 0.1 and 1.0}$$

5.3.2 Total Cost Factor for the Profile Cache

In the previous section, we found the cost factor for a single profile (Equation 5.5), using which we find the total cost factor of the profile cache. A profile cache has a number of profiles. The cumulative effect of the cost factors of all the profiles in the profile cache needs to be considered.

$$Total\ Cost\ Factor = \sum_{i=1}^{n} (N_{Est_i} - N_i) * S_{result_i} \tag{5.6}$$

The size of result (S_{result}) can vary from one profile to another in a profile cache. Therefore, the size of the result has to be considered to give the right leverage to each profile. The profile cache is moved if the *Total Cost Factor* (Equation 5.6) is positive.

5.4 Summary

Profile cache is moved based on the *Total Cost Factor* (Equation 5.6) that has been derived using data transmission costs. By doing this, best of both the approaches (keeping the profile cache on HA and moving it every time the user moves) is inherited. Hybrid approach has low bandwidth usage and low result notification cost along with a higher profile cache availability time. Moreover, similar profile can be grouped so as to share common computation.

CHAPTER 6

PERFORMANCE EVALUATION

Simulation experiments are conducted to evaluate the performance of InfoAccess and a prototype has been developed to assess the feasibility of InfoAccess. This chapter explains the need for using both the simulator and the prototype, experimental set up for the simulations, experimental analysis and some of the issues in developing the prototype.

6.1 Need for both the simulator and the prototype

InfoAccess is a distributed system that is designed for the mobile networking infrastructure. Hence, the basic assumption is that the mobile networking infrastructure is available. A network simulator not only provides the infrastructure that is needed for InfoAccess but also enables experimentation with varying number of mobile nodes at different speeds. Thus, a simulator is needed to validate the cost factor that is used for migration of profile cache. On the other hand, InfoAccess accesses information from real world webpages and database, which is not quite possible using a simulator. Thus, the functionality of accepting the profiles from the user, processing them efficiently using query graphs, grouping similar profiles has been developed as a prototype.

6.2 Simulation

The simulation experiments are conducted using the ns2 simulator. The mobile IP module of the ns2 simulator has been extended to provide some of the functionalities of the InfoAccess server. Profile Manager and Profile Cache Manager have been implemented.

The profile manager module has been implemented in the home agents of the mobile clients. The mobile client submits new profiles to the profile manager. The profile manager sends it to the profile cache manager that is currently executing the profile cache of that particular mobile client. The number of profiles submitted by each mobile client varies between 1 and 10. The profile size of each of the profiles varies between 1KB and 4KB. The result size of a profile varies between 200 bytes and 200KB. The periodicity of a profile varies between 10 simulation seconds and 50 simulation seconds. The number of profiles for a mobile client, the size of the profile, the size of the result for a profile and the periodicity of a profile are generated randomly using a random number generator with uniform distribution.

The profile cache manager receives the profiles from the profile manager. The periodic execution is simulated by using timers in ns2. A timer is associated with each and every profile that is being executed. The timer goes off every time period of the profile. When the timer goes off, the execution of the profile is simulated by incorporating certain amount of delay. Then, the result is sent to the mobile client.

The "decision module" of the profile cache manager has been implemented. When the mobile client moves from one BS to another, handoff occurs. When the handoff occurs, the decision module is notified. The decision module decides whether or not to move the profile cache based on the total cost factor. The total cost factor of the profile cache is calculated using the Equation 6.1.

$$Total\ Cost\ Factor = \sum_{i=1}^{n} (N_{Est_i} - N_i) * S_{result_i} \tag{6.1}$$

N_{Est_i} is the estimated number of data pushes and is calculated using Equation 6.2
N_i is the cutoff value for the profile i and is calculated using Equation 6.3
S_{result_i} is the size of the result of profile i

$$N_{Est_i} = (\alpha * T_{cur} + (\alpha - 1) * T_{avg}) \ /\ periodicity \tag{6.2}$$

T_{cur} *is the time that the mobile client has stayed in the current BS*

T_{avg} *is the average time that the mobile client has stayed in a BS*

α *is the leverage given to current data*

In our experiments, current data is given a leverage of 20% whereas historical data is given a leverage of 80%. Thus, α=0.2 is used in all the experiments.

$$N_i = (S_{profile_i} + S_{result_i}) \ / \ S_{result_i} \tag{6.3}$$

$S_{profile_i}$ *is the size of the profile i*

If the total cost factor of the profile cache calculated using Equation 6.1 is positive, then the profile cache is deleted from the old BS, the profile cache is transferred across the wired network to the new BS and the profile cache is added to the new BS.

6.2.1 Experimental Setup

Experiments were conducted for the following approaches and various parameters were compared to see which approach performs the best.

1. Profile cache on HA - In this approach, the profile cache is kept stationary at the HA.
2. Profile cache on FA - In this approach, the profile cache is moved to every BS that the mobile client visits. Therefore, the profile cache is maintained on the FA that is currently servicing the mobile client.
3. Hybrid approach - The cost factor shown in Equation 6.1 is used to dynamically decide whether or not to move the profile cache.
4. 60% (MobileIQ) - In this approach, the profile cache is moved 60% of the time randomly. To the best of our knowledge, MobileIQ is the only existing scheme that supports migration of profile cache. But, in MobileIQ, profile cache is migrated only

if the user requests for migration. To simulate the effect of migration in MobileIQ, we have assumed that the user requests migration of profile cache 60% of the time.

The performance of the proposed scheme (Hybrid approach) is evaluated in terms of the following parameters

- Bandwidth Usage - Bandwidth usage is the total bandwidth used over the wired network.
- Percentage of data loss - This parameter denotes the small percentage of data lost when the profile cache is being transferred.

Experiments were conducted using the parameter settings shown in the Table 6.1

Table 6.1 Parameter Settings

Parameters	**Values**
Simulation time	1000 secs
Number of BSs	7
Number of Mobile Clients	70 - 140
Speed of Mobile Clients	10m/s - 40m/s
Number of profiles per Mobile Client	1 - 10
Size of each profile	1KB - 4KB
Size of result for each profile	200bytes - 200KB
Periodicity of each profile	10secs - 50secs

6.2.2 Experimental Results

6.2.2.1 Experiment 1

The total bandwidth usage for data pushes over a period of 100 seconds is calculated and plotted as shown in the Figure 6.1. The worst case for bandwidth usage is when the cache is on the HA. The best case is when the cache is on a FA. “Hybrid” and “60% (MobileIQ)” approaches give average case bandwidth usage. Even though, bandwidth

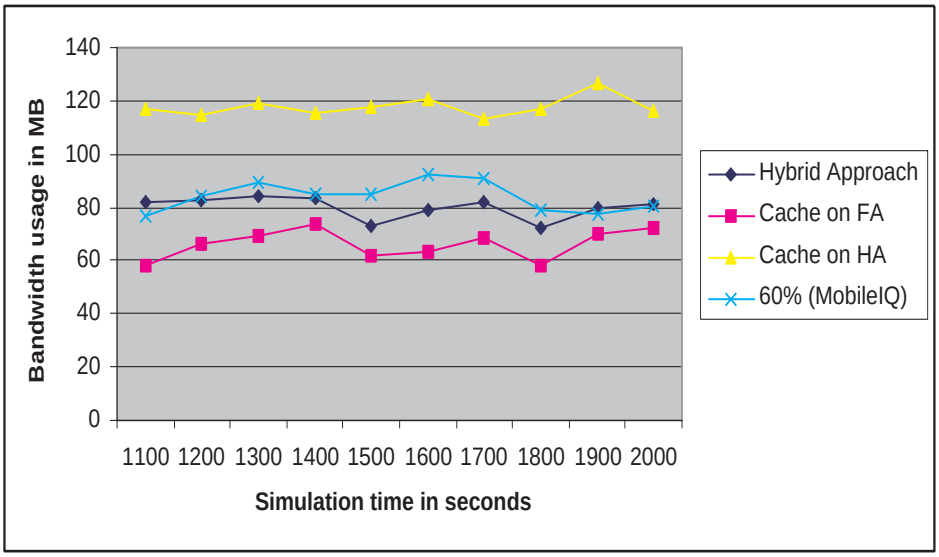


Figure 6.1 Bandwidth Usage Vs Time

usage is the least when the cache is on FA, this approach has higher percentage of data loss as shown in experiments 2 and 3.

6.2.2.2 Experiment 2

This set of experiments show the behavior of all the four approaches in terms of total bandwidth usage and percentage of data loss as the number of mobile nodes increase. The maximum speed of the mobile nodes is set to 20m/s and the experiments are conducted for 70, 100, 120 and 140 mobile nodes.

The bandwidth usage for the approaches "Cache on FA" and "Hybrid Approach" differ by a small percentage (10%-15%) as shown in the Figure 6.2. However, "Hybrid approach" performs well as compared to "Cache on FA" in terms of percentage of data loss. The percentage of data loss in "Hybrid appraoch" is 40%-60% lower than the "Cache on FA" approach as shown in the Figure 6.3. Even though "Cache on HA" approach has 0% data loss, it has the highest bandwidth usage and also gives no opportunity

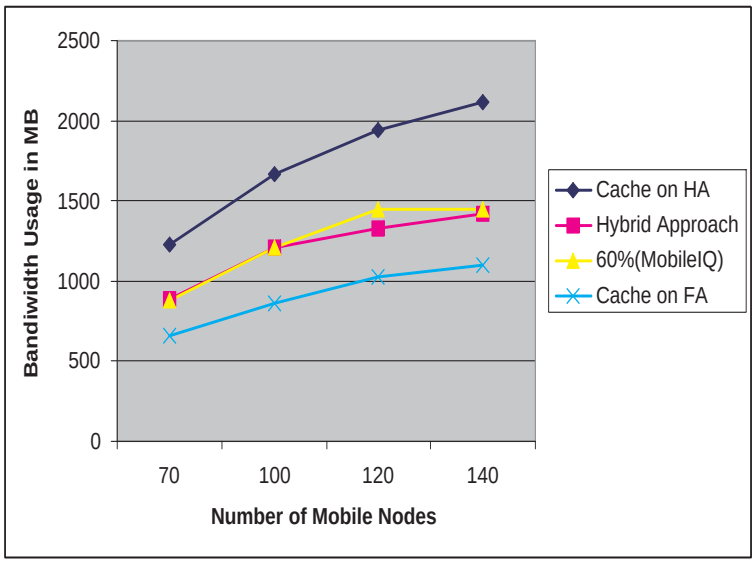


Figure 6.2 Bandwidth Usage Vs Number of Mobile Nodes

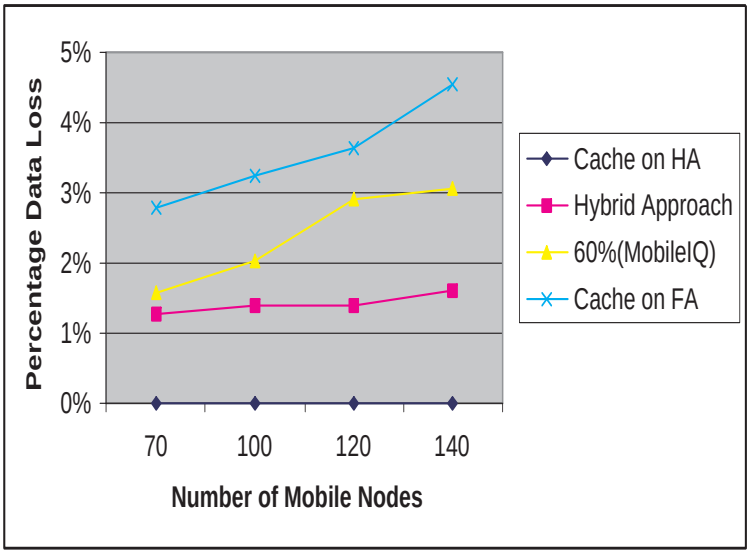


Figure 6.3 Percentage of Data Loss Vs Number of Mobile Nodes

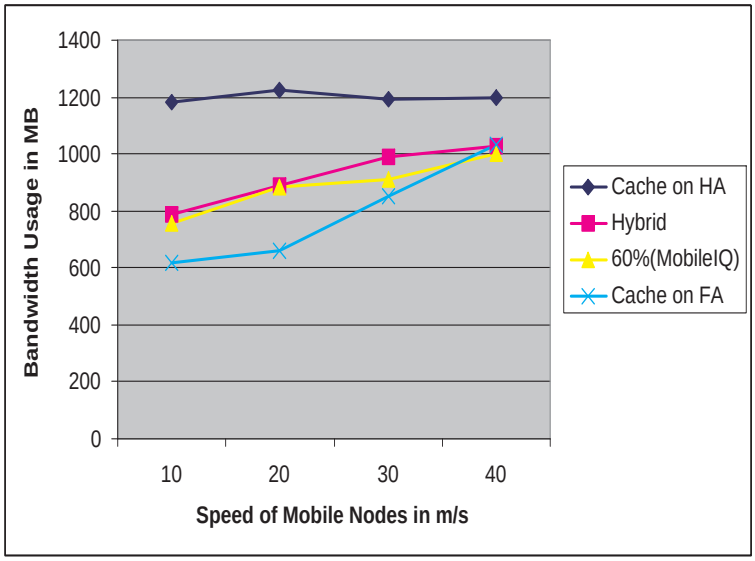


Figure 6.4 Bandwidth Usage Vs Speed of Mobile Nodes

for grouping of profiles. "Hybrid approach" performs slightly better than the "60% (MobileIQ) approach" in terms of bandwidth usage. The percentage data loss in "Hybrid approach" increases slightly with increase in number of mobile nodes, whereas for the "60% (MobileIQ) approach", the increase is dramatic.

6.2.2.3 Experiment 3

This set of experiments show the behavior of the three approaches in terms of total bandwidth usage and percentage of data loss as the maximum speed of mobile nodes increase. These experiments are conducted for 70 mobile nodes and the maximum speed for the mobile nodes is set at 10m/s, 20m/s, 30m/s and 40m/s.

Here too, we can observe the same behavior as was seen in Experiment 2. In terms of bandwidth usage (Figure 6.4), "Cache on FA" approach does the best. In terms of percentage of data loss (Figure 6.5) , "Cache on HA" approach does the best. "Hybrid approach" and "60% (MobileIQ) approach" give average case bandwidth usage. But, the

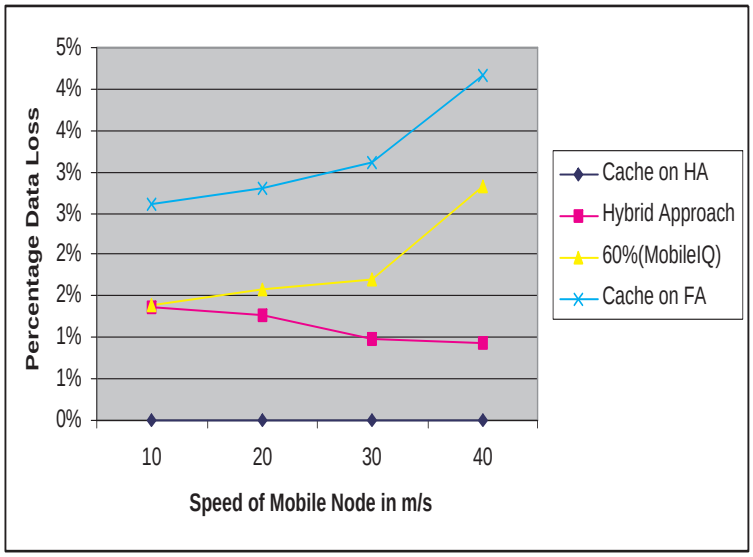


Figure 6.5 Percentage of Data Loss Vs Speed of Mobile Nodes

percentage data loss increase with the speed of the mobile nodes for both "Cache on FA" and "60% (MobileIQ)" approaches.

As the maximum speed of mobile nodes increase, bandwidth usage of the "Cache on FA", "60% (MobileIQ)" and "Hybrid" approaches increase(Figure 6.4) showing that the migration of profile cache become less beneficial. This is because the mobile node is not connected to the same server long enough to make the migration beneficial. In "Cache on FA" and "60% (MobileIQ)" approaches profile cache is moved to the new sever irrespective of the speed of a mobile node or the amount of time a mobile node stays connected to the server. As the mobile nodes move faster, they change more number of servers. Therefore, the profile cache is migrated many times leading to increase in data loss as the speed of mobile nodes increase. However, in "Hybrid" approach, as the speed of mobile nodes increase, the profile cache is moved for lesser number of times. Therefore, there is a decrease in data loss as shown in the Figure 6.5.

6.2.3 Conclusion

The results from the experiments performed show that there is no one approach that performs well in terms of both the parameters - bandwidth usage and percentage of data loss. There is a trade off involved in terms of data loss when the profile cache is moved. But, having the cache close to the user has benefits of low bandwidth usage and grouping possibilities. From the experiments, it can be concluded that "Hybrid Approach" performs the best among the three approaches, when all the parameters are considered.

6.3 Prototype

A stand alone application has been developed to implement the functionalities of accepting the profile, building query graphs for the profiles, grouping the profiles and providing the results at right intervals to the user.

6.3.1 Implementation of ECA rules

A *profile* object is created for every profile that is submitted by the user. The *profile* object maintains all the control parameters and data parameters of the profile such as the lifespan, the notification interval, the data source, keywords, filter conditions etc. The *profile* object is accessed to fetch any of these parameters during runtime.

In InfoAccess, LED is used to generate start and end events that correspond to start and end lifespan of profiles. ECAAgent class in LED contains the API to be used for generating events. The ECAAgent instance stores the names of all events and their associated event handles. The handle to ECAAgent is obtained as shown below:

```
import sentinel.led.*;
ECAAgent myAgent = ECAAgent.initializeECAAgent();
```

With the API provided by ECAAgent, the user can create class level and instance level

primitive events, composite events and define rules on those events. The table 6.2 shows the methods used in creation of events and rules.

Table 6.2 Methods of ECAAgent Class

Method	Return Type	Description
CreatePrimitiveEvent	EventHandle	This method creates a primitive event of instance level
CreateCompositeEvent	EventHandle	This method creates a composite event of instance level
CreateRule	EventHandle	This method creates an instance level rule on the specified object instance
RaiseBeginEvent	static void	This method raises an event at the beginning of the method

The start and end events are created using the "CreatePrimitiveMethod" method. Periodic events are created using "CreateCompositeEvent". Rules on these events are created using "CreateRule" method. "RaiseBeginEvent" is used to raise events. Start and end events are created for each and every profile in the system. Rule associated with start event adds a profile to the query graph. The rule corresponding to the end event deletes a profile from the query graph. Rule associated with the periodic event processes the profile and notifies results to the user.

6.3.2 Implementation of Query Graphs

The key classes used in the implementation of query graphs are shown in the Figure 6.6. Primitive result computation is done by filter nodes. Composite result computation is done by composite nodes - AND, OR, NOT, UNION and INTERSECTION. The class *Node* is an interface, the class *Composite* is an abstract class and the other classes are concrete classes. The *Node* interface has an *insertProfile* method that is implemented

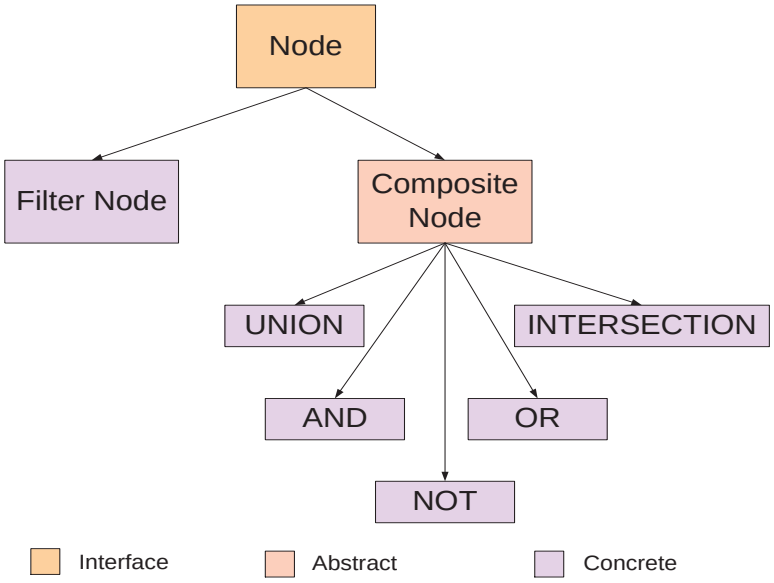


Figure 6.6 Query Graph Classes

by all the concrete classes shown in the Figure 6.6. The *insertProfile* method implements the addition of profiles to the nodes. All the nodes have linked list associated with them called the *Subscriber List*. Subscriber List contains references to all the nodes that are one level above them and are connected to them. Subscriber List also contains references to profile objects. The nodes are associated with *result* object that is propagated to all subscribers.

6.3.2.1 Data Source Node

The *dataSource* Node is the class that represents the URL representing the webpage or a table in the database. A list of data sources (URLs or tables) and their corresponding references are maintained in a hash table named the *dataSource List*. The table 6.3 shows the methods of the class *dataSource* node. The Subscriber List of the dataSource node contains references to the *information* node objects.

Table 6.3 Member functions of DataSource Node

Method	Description
insertFilterNode	Inserts filter nodes to the subscriber list. Returns true if properly inserted
getNumberOfProfiles	Returns the number of active profiles fetching the results from this data source.
propagate	This method is invoked when data corresponding to this data source is fetched. The data is propagated to all the subscribers.

6.3.2.2 Filter Nodes

Filter nodes of the query graph are responsible for computing the customized data required by the user. In case of webpage data source, filter nodes check for filter conditions (if any) specified on the information node. In case of database data source, filter nodes extract the keywords (project attributes) from the tuples that are sent to the filter node. The *filter* node object contains references to

- Subscriber list containing references to profiles and composite nodes.
- Producer (dataSource Node)
- Result computation algorithms

The table 6.4 shows the methods of the *filter* node.

6.3.2.3 Composite Nodes

The composite node receives result from one (in case of NOT and INTERSECTION) or two (in case of AND, OR and UNION) nodes. These nodes are called the children of composite nodes. Composite nodes create proxy profiles at their children nodes and add these proxy profiles in to the subscriber list of children nodes. Thus, children nodes propagate results to composite nodes, i.e., the proxy profiles at children nodes invoke the method *propagate* of composite nodes.

Table 6.4 Member functions of Filter Node

Method	Description
insertProfile	This method is invoked when a new profile is inserted.
computeResult	This method is invoked when new data arrives from the producer. Returns results computed from the data
propagate	This method is invoked after the result computation. Results are propagated to all the subscribers. Before propagating results to profiles, filter conditions specified on this filter node are checked in case of webpage data source and keywords (project attributes) are extracted from the result for database data source.

The table 6.5 shows the propagate method of all the composite nodes for a webpage data source. The propagate method has the prototype *propagate(Result res, Boolean flag)*. For a webpage data source, the result propagated by the child node is the information corresponding to a keyword and the flag is a boolean value which corresponds to the filter conditions.

Table 6.5 Propagate Method of composite nodes for a webpage data source

Composite Node	Description
NOT	The subscriber is notified only if the condition is not satisfied, i.e., the flag is false
AND	The merged result from both the children nodes is notified only if both the conditions (flags) are true. If there is no filter condition associated with the child node, the condition is assumed to be satisfied (true).
OR	The result of the child node with the flag true is notified to the subscriber.

The table 6.6 shows the propagate method of all the composite nodes when the data source is a database. The propagate method has the prototype *propagate(ResultSet result)*. For a database , the result propagated by the child nodes are the tuples that

satisfy the filter condition (predicate). The methods of the class *ResultSetMetaData* are used to obtain the schema of the result (tuples) passed by the children nodes. Various operations can be performed on the tuples using the methods provided by *ResultSet* and *ResultSetMetaData* classes.

Table 6.6 Propagate Method of composite nodes for a database data source

Composite Node	**Description**
UNION	The tuples obtained by both the children nodes are merged and then the subscriber is notified.
INTERSECTION	The filter condition specified in this node is applied on the tuples obtained from the child node and the resultant tuples are notified to the subscriber.

6.3.3 Implementation of Grouping Data Structure

The *group* class representing the grouping data structure maintains a handle to the periodic event of the group, periodicity and a table called the filter table. In case of webpage data source, the filter table is indexed by keyword. In case of database data source, the filter table is indexed by filter condition. For each element in the table, a linked list of all the profiles that need the element and the count is maintained. The count gives the number of time periods after which the information has to be sent to the user. The important methods of this class are shown in the table 6.7.

Thus, ECA paradigm has been used to support scheduled queries and periodic queries in InfoAccess. The prototype also implements query graphs and grouping data structure that allows optimized computation of results.

Table 6.7 Member functions of Group

Method	**Description**
doesBelong	This method returns true if the input profile can be merged with the existing group.
insertProfile	Inserts the profile in the group. It also sets the periodicity of the group to an appropriate value.
getProfileCount	This method returns the number of profiles in the group.
propagate	This method propagates the result for the filter node encapsulated with the profiles that require the information corresponding to this particular filter node. It sends the results to the data source node.

CHAPTER 7

RELATED WORK

Considerable amount of work has been done in the area of information retrieval for mobile clients. The client-proxy-server model has begun to feature in many mobile applications in the mobile computing world. The main goal here is to overcome the constraints posed by limited bandwidth and resource poor mobile devices. Proxies enhanced with functionalities, also known as *Intermediaries* are deployed on the wired network to alleviate the problems. Research and development on intermediary software have produced a variety of systems. This chapter discusses some of the systems that address the issues of information retrieval for mobile clients.

7.1 Mowgli

The Mowgli System [7] consists of two mediators, the Mowgli Agent and the Mowgli Proxy located on the mobile node and the base station respectively. They use the Mowgli HTTP protocol to communicate with each other, which reduces the number of trips between the client (mobile node) and the data server. Mowgli WWW reduces the data transferred over wireless link in three ways: data compression, caching and filtering. As this is one of the earliest systems, there is no notion of personalization or customization of data. Filtering is based on the maximum size of the document specified by the user. If an incoming document exceeds the maximum size, it is either sent without the embedded images or it is not sent at all. We believe that this kind of filtering is rather naive and should be based on user interests. Also, there is no notion of periodic queries that is very

important in a dynamically changing environment. Mowgli is fairly limited in the data sources it can handle as it supports only web servers and not traditional databases.

7.2 WebExpress

WebExpress [8] is a client/intercept-based system that optimizes web browsing on resource-poor clients connected over wireless connections. It dispatches the *Server Site Intercept (SSI)*, on the wired network and the *Client Site Intercept (CSI)* on the end user's mobile node. The two agent run a stripped down version of HTTP protocol, optimized for wireless communication. Both agents provide caching of content, and run a simple differencing protocol between the client-site and the server-site cache to optimize the provision of dynamic content. Each new request to a particular URL may well result in a different response. Essential to differencing is caching a common base object on both CSI and SSI. When a response is received, the SSI computes the difference between the base object and the response and then sends the difference to the CSI. The CSI then merges the difference with its base form to create the browser response. This technique is used determine the difference between HTML documents.

The SSI and CSI are specific to the type of application. The drawback of this model is that every application that is to be accessed requires development work , for the implementation of CSI/SSI pair, in the server and client sites. InfoAccess provides a generalized scheme for processing data from each type of data source and is not specific to application. Therefore, development work is needed in the InfoAccess server for every type of data source rather than every type of application. Here too, periodic queries are not supported and grouping of queries has not been considered.

7.3 iMobile

The iMobile project [9] of AT&T research is a proxy based system that provides information retrieval for mobile clients. The iMobile proxy maintains user and device profiles, accesses and processes internet resources on behalf of the user, keeps track of user interaction and performs content transformations according to device and user profiles. The architecture of iMobile is established upon the infrastructure of iProxy [18], that offers a suite of archiving, retrieval and searching services. iMobile consists of three main abstractions: devlets, infolets and applets.

A devlet is an agent-abstraction for supporting the provision of iMobile services to different types of mobile devices connected through various access networks. The output data is encoded in a MIME type appropriate for devlet's terminal device before sending the data to the terminal device. The infolet abstraction provides a common way of expressing the interaction between the iMobile server and the data sources. Different data sources export different interfaces to the outside world: JDBC and ODBC for databases, webpages for WWW, IMAP for email servers, etc. Finally, an applet is a module that processes and aggregates content retrieved by different sources and relays results to various destination devices. At the core of an iMobile server resides the "let engine", which registers all devlets, infolets and applets, receives commands from devlets, forwards them to the right infolet or applet, transcodes the result to an appropriate terminal-device format and forwards it to the terminal device via the proper devlet.

InfoAccess is different from iMobile in the following ways. User profiles in InfoAccess is an extension of user profiles in iMobile. In iMobile, user profile specifies just the type of the data source and does not actually specify user interests. In InfoAccess, user profile specify the data source, the keywords the user is interested in and the filter conditions that are to be applied. Thus the level of personalization provided by iMobile is low as compared to the level of personalization in InfoAccess. iMobile archives data that

has been previously accessed by the user. the emphasis is more on accessing information from the archives than from accessing new data. InfoAccess caches the most recent result and gives more emphasis on accessing new information from the data servers.

7.4 MobileIQ

MobileIQ [4] is a distributed system that offers personalized mobile information access from the WWW in a wireless network environment. MobileIQ is actually an extension of W^3IQ [19], a proxy-based system that aggregates personalized web content on the wired side based on user profiles, stages it at the proxy and provides it to the user when they connect to the proxy. MobileIQ allows the user to specify requests as queries, rather than URLs and takes up the burden of locating the right information on behalf of the user. The URLs returned as results are the ones that the system feels will contain the information user requested. Users then prioritize the URLs provided by the system by rating them, depending on their perception of the relevance of the URL. The user feedback allows the system to infer a semantic match between the user's query terms and the information in the document. Based on the user feedback, the system creates a user's profile which is used to control the information he/she would receive to future queries. As it is based on the feedback, first-time users or infrequent users benefit less from the system. A lot of unnecessary data is provided to such users. This not only increases the bandwidth usage across the wireless link but is also annoying to users as they are flooded with unwanted information. MobileIQ has to be trained over a period of time to retrieve right personalized information for a particular user. The user will have to give enough feedback for the system to learn the user's specific requirements.

Users submit queries in one of the many predefined categories and rate the results on a scale of 1 to 10. Clustering (grouping) is performed only when a user has submitted considerable number of queries and rated enough information in a category. In InfoAccess,

queries are grouped based on the data sources which will increase the number of queries that will be grouped. When a user moves from one W^3IQ server to another, the user profile will be transferred to the nearest W^3IQ server only if the user requests for the transfer. This is cumbersome for the user as the user has to know when he/she has entered a new network. In InfoAccess, profiles and cached results of a user are migrated based on the transmission cost. Migration is transparent to the user. Also, in MobileIQ, computation for grouped queries is always done on the home agent. Periodic queries are not supported in MobileIQ.

7.5 eRace

The *extensible Retrieval, Annotation and Caching Engine* (eRACE) [10] is a middleware infrastructure designed to support the development and deployment of intermediaries on Internet. eRACE is modular, configurable and distributed proxy infrastructure that collects information from heterogeneous Internet sources and protocols according to XML-encoded eRACE profiles, registered within the infrastructure. Collected information is stored in a software cache for further processing, personalized dissemination to subscribed users, and wide-area dissemination on the wireline or wireless Internet.

eRACE is similar to InfoAccess in the sense that it supports personalization by enabling the registration, maintenance and management of personal profiles representing the interests of individual users. Its structure allows the easy customization of service provision according to parameters, such as information access model of choice (pull or push), client-proxy communication (wireline or wireless) and client-device capabilities (PC, PDA, mobile phone). Profiles expressed in XML requires the user to understand XML and specify queries using XML whereas profiles expressed in text is easy for users to populate.

At the core of eRACE lies WEBRACE, the agent-proxy that deals with information sources on the WWW, accessible through the HTTP protocols. WEBRACE consists of a crawler, an object cache storing multiple versions of retrieved resources, and an annotation engine that indexes collected resources, executes user-queries, and produces user-alerts encoded in XML.

eRACE provides the notion of periodic queries. A Unified Resource Description (URD) is an XML-encoded data structure, which is part of an "eRACE request", and describes source information for Internet sources monitored by eRACE. In essence, each URD represents a request to retrieve and process content from a particular Internet source on behalf of a particular user or service. A "Request Scheduler" scans continuously a database of URDs and schedules URDs for execution. This is a naive method of scheduling the execution. In InfoAccess, active capability has be made use to asynchronously schedule execution of queries.

The authors of eRACE have recognized the benefits of keeping the query execution close to the user by mentioning it as future work in eRACE. InfoAccess keeps the execution close to the user by dynamically migrating the profile cache thereby decreasing the bandwidth usage on the wired network.

7.6 User Profiles

User profile form the basis of many types of information delivery systems ranging from information filtering applications to the personalization of the content on the WWW. Profiles are excellent mechanism to capture interests of individual users. There has been considerable amount of work done in this area.

User profiles for web-based applications(Yahoo, PointCast [20]) are typically fairly simple allowing the user to specify particular categories (channels) of information they are interested in receiving. This approach to building information dissemination systems

typically uses a *walled garden* approach in which the data that can be delivered to the user is restricted to that found on specific content sites. The user should be able to specify his interests in a more general form and should be able to specify them for various different data sources.

User profiles for text-based data have been extensively investigated in the context of information filtering and selective dissemination of information [21]. The systems in these areas use techniques from the information retrieval (IR) world for filtering unstructured text-based documents [22]. In general, IR systems use either a boolean model or a similarity-based model. In the boolean model, a user profile is constructed by combining keywords with boolean operators (e.g., And, Or, Not), and an "exact match" semantics is used - a document either satisfies the predicate or not. Similarity-based models use a "fuzzy match" semantics in which the profiles and documents are assigned a similarity value (typically based on a vector space model [23]). A document whose similarity to a profile is above a certain threshold is said to match that profile. The Stanford Information Filtering Tool (SIFT) [24] is a well-known content-based text filtering system for Internet news articles. XML has also been used to define profiles and XFilter [25] is a recent example of such a filtering system. Profiles for data recharging has been proposed in [26]. Here, the profiles have two sections - a DOMAIN section and a UTILITY section. A profile's DOMAIN section specifies the data objects of interest to the user and the UTILITY section specifies the utility function that depicts the relative worth of data objects specified in its profile domain. Although useful in data recharging context, utility functions become burdensome for the user in information retrieval systems as the user needs to declaratively specify data objects and provide a utility function for the data objects.

7.7 Distinct features of InfoAccess

eRACE, MobileIQ, iMOBILE and most other systems for information retrieval for mobile clients process user queries on the servers to which the profiles were submitted. Result notification costs increase as the users move farther away from those servers. InfoAccess migrates the execution of profiles (queries) to follow the movement of the user. This not only reduces the result notification cost but also makes the execution of profiles location-aware. MobileIQ addresses this issue to certain extent. But, the query execution is moved only when the user requests for movement. This is burdensome on user's part and therefore, the movement should be transparent to the user.

Grouping of queries is another area that has not been exploited in many of the systems. As the execution of profiles is kept geographically close to the user in InfoAccess, number of profiles that are querying for local information are large. This gives us the opportunity to optimize computation by grouping similar queries.

We also address the issues with respect to scheduled queries and periodic queries. Active capability can be used to achieve this. Active technology to make databases active has been well developed. Active database systems are based on rule definition, event detection, and action execution. They add the additional functionality to recognize specific situations (external situations) and react to them in contrast to the traditional passive databases. Such databases not only detect composite events but also incorporate coupling and parameter modes for added capability. Active databases such as HiPAC, Sentinel, Ariel [27, 28, 29] have incorporated the active capability into the database. However, the usage of this capability has been very limited in web-based systems. To the best of our knowledge, active capability and the ECA paradigm have not been used in the area of information retrieval for mobile clients.

Most of the systems address issues relating to information retrieval from WWW. However, databases form a major source of information and the mobile user should be

able to access information from databases. Therefore, we support information retrieval from WWW and databases. The profile specification used in InfoAccess allow the user to schedule the execution of the profile, specify the data of interest either in WWW or databases and specify periodic execution of the same query.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

In this thesis, we have proposed InfoAccess, a scheme that offers personalized, location-aware information access in mobile environments using a distributed approach. InfoAccess has been proposed with the intent of providing mobile users the ability to schedule traditional, location-dependent and periodic queries. The mobile users are allowed to retrieve information from disparate sources such as WWW and relational databases. We have also looked into the issues of optimizing the bandwidth usage across the wired network and the issues of optimizing result computation on the servers.

To facilitate the user to describe his/her interests, profile specification language has been proposed. The user can not only specify his/her interests in terms of data source, keywords and filter conditions but also specify the start time, end time and the frequency at which the user wants the profile to be processed.

The profile cache, which is a collection of profiles and the cached results for a mobile user, is maintained on a server close to the mobile client. The migration of profile cache is dynamic and is based on the data transmission cost. The migration is transparent to the mobile user. Simulation experiments have been conducted to show that dynamic migration based on the transmission cost reduces bandwidth usage across the wired network and has lower percentage of data loss.

Active capability has been used to process profiles submitted by the user. Query graphs are constructed such that similar profiles share computation. Grouping data structure has been used to keep track of profiles that belong to the same group based

on periodicity. A prototype has been developed to group the profiles and process the profiles efficiently.

InfoAccess can be extended to support device profiles along with user profiles. Based on the device profiles, the system can predict the capabilities of the mobile device and transform results to a format that can be handled by the device. Search engines and mining techniques can be used to retrieve data of interest instead of user having to specify the data source. General schemes for retrieving information from HTML pages can be explored.

REFERENCES

[1] T. Imielinski and B. R. Badrinath, "Mobile wireless computing: challenges in data management," *Commun. ACM*, vol. 37, no. 10, pp. 18–28, 1994.

[2] G. Forman and J. Zahorjan, "The challenges of mobile computing," *IEEE Computer*, vol. 27, pp. 38–47, 1994.

[3] M. Dunham, A. Seydim, and V. Kumar, "Location dependent query processing," 2001.

[4] P. Chandrasekaran and A. Joshi, "Mobileiq: A framework for mobile information access," *Proceedings of the thrid International Conference on Mobile Data Management(MDM'02)*, 2002.

[5] B. C. Housel and D. B. Lindquist, "Webexpress: a system for optimizing web browsing in a wireless environment," in *Proceedings of the 2nd annual international conference on Mobile computing and networking*. ACM Press, 1996, pp. 108–116.

[6] M. D. Dikaiakos, "Intermediary infrastructures for the world wide web," *Comput. Networks*, vol. 45, no. 4, pp. 421–447, 2004.

[7] M. Liljeberg, M. Kojo, and K. Raatikainen, "Enhanced services for world-wide web in mobile wan environment," University of Helsinki, Tech. Rep. C-1996-28, April 1996.

[8] B. C. H. G. Samras and D. B. Lindquist, "Webexpress: A client/intercept based system for optimizing web browsing," *ACM/Baltzer Journal of Mobile Networking and Applications(MONET)*, pp. 419 – 431, 1998.

[9] C.-H. H. Rao, Y.-F. R. Chen, D.-F. Chang, and M.-F. Chen, "imobile: a proxy-based platform for mobile services," in *Proceedings of the first workshop on Wireless mobile internet*. ACM Press, 2001, pp. 3–10.

[10] M. Dikaiakos and D. Zeinalipour-Yazti, "A distributed middleware infrastructure for personalized services," Department of Computer Science, University of Cyprus, Tech. Rep. TR-01-4, Dec. 2001. [Online]. Available: citeseer.ist.psu.edu/dikaiakos01distributed.html

[11] S. Chakravarthy, "Design of sentinel: An object-oriented dbms with event-based rules," *Information and Software Technology*, vol. 36, no. 9, pp. 559–568, 1994.

[12] R. Dasari, "Events and rules for java: Design and implemenation of a seamless approach," Gainesville., 1999.

[13] S. Seshan, M. Stemm, and R. Katz, "Spand: Shared passive network performance discovery," in *1st Usenix Symposium on Internet Technologies and Systems (USITS'97)*, 1997.

[14] M. L. Anwar, E. and S. Chakravarthy, "A new perspective on rule support for object-oriented databases," *ACM SIGMOD Conf. on Management of Data*, pp. 99–108, 1993.

[15] W. Tanpisut, "Design and implementation of event based subscription/notification paradigm for distributed environments," Arlington, TX, 2001.

[16] S. Chakravarthy and D. Mishra, "Snoop: An expressive event specification language for active databases," *Data and Knowledge Engineering*, vol. 14, no. 10, pp. 1–26, 1994.

[17] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim, "Composite Events for Active Databases: Semantics, Contexts, and Detection," 1994, pp. 606–617.

[18] H. C.-H. Rao, Y.-F. Chen, and M.-F. Chen, "A proxy-based personal web archiving service," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 1, pp. 61–72, 2001.

[19] A. Joshi, C. K. Punyapu, and P. Karnam, "Personalization asynchronicity to support mobile web access," in *Workshop on Web Information and Data Management*, 1998. [Online]. Available: citeseer.ist.psu.edu/joshi98personalization.html

[20] S. Ramakrishnan and V. Dayal, "The pointcast network," *ACM SIGMOD Conf.*, 1998.

[21] P. W. Foltz and S. Dumais, "Personalized information delivery: An analysis of information filtering methods," *Commun. ACM*, vol. 35, no. 12, pp. 51–60, 1992.

[22] N. J. Belkin and B. Croft, "Information filtering and information retrieval: Two sides of the same coin?" *Commun. ACM*, vol. 35, no. 12, pp. 29–38, 1992.

[23] Y. C. Salton, G. and A. Wong, "A vector space model for information retrieval," *Commun. ACM*, vol. 18, 1975.

[24] T. Yan and H. Garcia-Molina, "The sift information dissemination system," *ACM Transactions on Database Systems (TODS)*, vol. 24, no. 4, pp. 529 – 565, December 1999.

[25] M. Altinel and M. J. Franklin, "Efficient filtering of XML documents for selective dissemination of information," in *The VLDB Journal*, 2000, pp. 53–64. [Online]. Available: citeseer.ist.psu.edu/altinel00efficient.html

[26] M. Cherniack, M. Franklin, and S. Zdonik, "Expressing user profiles for data recharging," *Personal Commun., IEEE*, vol. 8, no. 4, pp. 32–38, 2001.

[27] e. a. Chakravarthy, S., "Hipac: A research project in active, time constrained database management," Xerox Advanced Information Technology, Cambridge, Tech. Rep. 89-02, 1989.

[28] e. a. Wells, D., "Architecture of an open object-priented database management system." *IEEE Computer*, vol. 25, pp. 74–81, 1992.

[29] E. Hanson, "The ariel project, in active database systems -triggers and rules for advanced database processing," *Morgan Kaufman Publishers Inc.*, pp. 63–86, 1996.

BIOGRAPHICAL STATEMENT

Anupama Mutt was born in Bangalore, India, in 1978. She received her Bachelor of Engineering in Computer Science degree from Mysore University, India, in 2000. She worked for Wipro Technologies, India as a Software Engineer from September 2000 to October 2001. In the Fall of 2002, she started her graduate studies in Computer Science at The University of Texas, Arlington. She received her Master of Science in Computer Science and Engineering from The University of Texas at Arlington, in December 2004. Her research interests include mobile computing, mobile databases, information retrieval and active databases.