An Architecture for Mobile, Distributed Application Delivery with Soft Real-time Constraints

Joel Jones, Susan Vrbsky, Jingyuan Zhang, Sibrabrata Ray

Department of Computer Science, University of Alabama
101 Houser Hall
Tuscaloosa, AL 35487 USA
(205) 348-6363

Abstract

The power of ubiquitous computing lies not just in constant access, but also in tailoring of information based upon location. In this paper we describe an architecture that supports tailoring of information and applications for their environment. This environment includes the mobile client device, its location, the available bandwidth, and any soft real-time constraints.

1 Introduction

We present here an architecture for the delivery of applications in a distributed, mobile environment with soft real-time constraints. Such a system has many practical uses, including emergency medical response, disaster relief work, and construction. This proposal includes motivation for the addition of quality of service (QoS) guarantees to the underlying system components (computer languages, compilers, networks, and databases), and to integrate these QoS guarantees throughout the overall system. As a result of this work, we establish a foundation for determining the fundamental characteristics such system components should have. This architecture provides a foundation for implementing mobile application delivery systems.

Email addresses: jones@cs.ua.edu (Joel Jones,), vrbsky@cs.ua.edu (Susan Vrbsky,), zhang@cs.ua.edu (Jingyuan Zhang,), sibu@cs.ua.edu (Sibrabrata Ray).

1.1 System Use Scenario

To clarify our system functionality, we illustrate using a prosaic example, consumer information. Imagine someone standing on a sidewalk in the North Beach neighborhood of San Francisco, California. The North Beach area is awash with Italian restaurants. A user with a mobile computing device (such as a PDA) issues the query "What are the closest Italian restaurants?" Our architecture defines the type of system needed to answer this query in a timely fashion.

Initially, the PDA uses wireless networking to find a query server to handle the query. The query server does not store applications, so it issues queries to application servers which contain the requested information, then merges the responses and transmits them to the PDA.

In our model, the ability to deliver both applications and information in a timely manner drives the remainder of the research questions. In this scenario, such programs might allow the user to make a reservation, view an animation of steaming lasagna, or initiate a phone call to the restaurant. This represents the "application delivery" portion of the system.

Since PDAs are heterogeneous, (i.e. the processor and other aspects of the PDA may vary from user to user), processor variability is handled by expressing the delivered applications as code for some sort of virtual machine, such as the Java Virtual Machine (JVM) [1]. This is the "mobile" portion of the system.

Our system has soft-real constraints, since an individual will not wait an indefinite amount of time to receive an answer to their query. As yet another constraint, if the PDA is located in a moving vehicle, information must be delivered before the user has moved out of the area. This is the "soft real-time" portion of the system.

Furthermore, businesses want to have control over the user-experience of their customers. A business will establish linkages to its applications on the query servers responsible for handling wireless networking in a given service area. Therefore, query servers will not have a unified database of applications, and must query multiple application servers to process a client's query. The services provided by a query server more closely align with those typically provided by an internet service provider or wireless telecommunications vendor. The services provided by an application server more closely align with those of an application hosting provider or by an enterprise itself. This dichotomy is the reason for splitting our infrastructure component into two pieces. These pieces could of course be combined into a single server if so desired. This contributes to the "distributed" portion of the system.

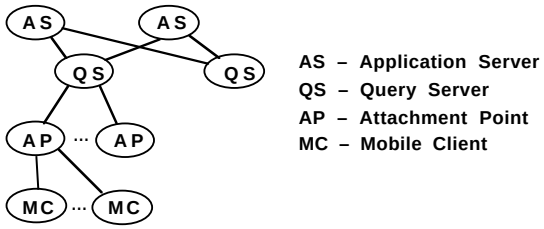

Fig. 1. System Architecture

1.2 Research Areas

Our proposed architecture is a framework in which interesting research with commercial applicability can occur. Already, there is a rapidly expanding infrastructure of IEEE 802.11 wireless networks, as well as low speed telephony networking [2–4]. The scenario outlined above exposes many interesting research issues that have not been addressed in this kind of integrated application. They include use of machine-independent performance profiles for soft-real time scheduling, distributed real-time spatial databases, and network bandwidth scheduling. Our architecture provides a framework for investigating these issues via its construction of a mobile, distributed application delivery system operating under soft real-time constraints.

1.3 Problem Scope

To provide a reasonable scope to build on, we limit the proposed system architecture. We make the following assumptions regarding our system model.

- Every mobile client is attached to a single attachment point (AP) at a time.
- As mobile clients move, the AP they communicate with may change (handoff). This should not result in a complete loss of service.
- Each AP may service multiple mobile clients.
- An AP does not communicate with other APs.
- Each AP communicates with a single query server (QS).
- Each query server will communicate with one or more APs.
- A query server will communicate with one or more application servers (AS).
- A query server will not communicate with other query servers.
- An application server will communicate with one or more query servers.
- Application servers will not communicate with each other.

We see a diagram of the components of the system in Figure 1.

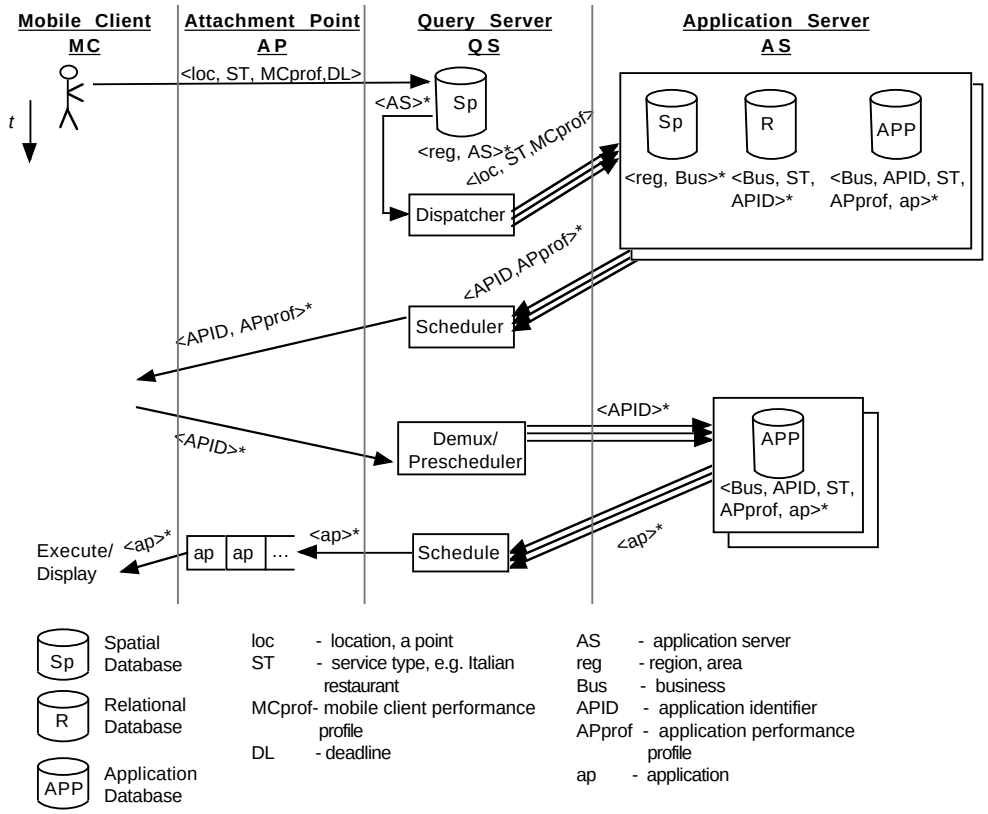

Fig. 2. Interaction Diagram for Simple Query

2 Example: Simple Query

In Figure 2 we see the interaction diagram for a simple location based query. Across the top of the figure, we see the four components of the system: the mobile client (MC), the network attachment point (AP), the query server (QS), and the application server (AS). The diagram shows the flow of information through the system to answer a query such as "Show me information on the closest Italian restaurants." In the rest of this section, we will present an overview of the steps necessary in answering this query.

2.1 Setup

Before the mobile client can even issue a query, it must perform resource discovery to find an attachment point and its associated query server. This is an area of much research and there are many deployed architectures for finding resources [5–8]. Once the attachment point has been found, then the additional steps of authorization and location determination must be done. Authorization is the determination of whether or not the mobile client (or equivalently, the associated user) is allowed to use the network. Location determination is the

process of ascertaining the location of the mobile client. There is an ordering of sources for location information, with earlier sources being more authoritative. First, the user must be allowed to override any of the later sources. Second, a GPS unit in the mobile client can give precise information on the location of the mobile client. The third location source is the determination of the mobile client's location by triangulation of several in-range attachment points. The location of the chosen attachment point is the fourth and last source of location information.

2.2 Determining Application Availability

The first step in application delivery is to determine which applications are available. A query is originated by the user with the mobile client. In this example, the query consists of the 4-tuple `<loc, ST, MCprof, DL>`, which communicates the location of the MC (`loc`), the type of service for which information is being requested (`ST`), here "Italian Restaurants", the performance profile of the mobile client (`MCprof`), and the deadline for delivery of the applications/documents (`DL`). [1] This query is sent to the query server (QS), that has a spatial database (Sp), which maps geographic regions (reg) to application servers (AS) that have information and applications for the corresponding region. The location is queried against the regions, and a set of application servers `<AS>*` is found. The query `<loc, ST, MCprof>` is sent to each of the matching application servers by the query server's dispatcher. The query server then sets up a scheduler to gather the responses and send them to the mobile client. The query sent from the QS to the AS is handled using the databases on the AS. A spatial database mapping location to regions is used to determine which businesses (Bus) serve the requested location. Then, for each business, a relational database is queried to find the application identifiers (`APID`) available for the requested business and service type. The APID is then used to query the application database to determine the applications performance profile (`APprof`). The resulting list of application IDs and application profiles `<APID, APprof>*` is sent back to the QS. The scheduler in the QS determines which results to return to the MC.

Results are rejected if they arrive from the AS too late to meet the deadline. Other results are rejected for being too lengthy to send or execute by examining the `APprof` and comparing it against the `MCprof` and information about the network congestion at the AP for the MC. The remaining results are sent to the MC: `<APID,APprof>*`.

To make this discussion more concrete, consider the earlier example. Assume that the user has a Handspring Treo 180 and that they have indicated that

[1] In general, queries are richer than just "find nearest service of type *foo*."

they are on foot. The initial query would have its location (`loc`) as the latitude and longitude of the street corner where the user is standing. The location was chosen by the user, as the Treo has no GPS and its cell phone network connection does not allow a fine-grained enough determination of location. The service type (`ST`) is Italian restaurant. The mobile client performance profile (`MCprof`) would state that its connection is at 9.6Kb, that there is voice phone, and that the display is monochrome, 4-bit grey-scale and 160×160 pixels. As a default setting, the deadline is set to be 30 seconds, based upon the indication that the user is on foot.

The are five potential applications available, of three different types. Three of them are phone reservation applications that require a mobile client equipped with a phone. Each of these applications is 3KB in size. Two of the restaurants with phone reservation applications, "The Stinking Rose" and "Il Fornaio", are located in North Beach. The third, "Frankie, Johnnie, and Luigi Too", is located in Mountain View. One application is a streaming, steaming lasagna animation and requires a connection speed of ≥48kb and a color display. The final application is an interactive menu and is 100KB in length.

There are three application servers. Application server #1 contains the phone application for "Il Fornaio", as well as the interactive menu. Application server #2 contains the phone application for "The Stinking Rose" and the lasagna video. Application server #3 contains the phone application for "Frankie, Johnnie, and Luigi Too."

The application query is sent from the mobile client to the query server. The query server does a containment query with the `loc` against its spatial database to determine which application servers cover the street corner. A query is formed from the `loc`, `ST`, `MCprof` fields and sent to application servers #1 and #2. Because application server #3 covers Mountain View and not North Beach, the query is not forwarded to it.

Application server #1 examines its spatial database to find a business whose region contains `loc`. It finds the businesses "Il Fornaio" and the one with the interactive menu. It forms a response to the query consisting of the application identifier `APID` and the application profile `APprof` for the two applications and sends it to the query server. Application server #2 does a similar spatial query and finds "The Stinking Rose" and the lasagna video. Examining the `MCprof` it sees that the lasagna video is not viewable on the `MC`. It sends the `APID` and `APprof` for "The Stinking Rose" to the query server.

The query server receives the responses from the application servers and examines their `APprof` fields. From the schedule of the bandwidth for the `AP`, the 100KB interactive menu application will not finish downloading before the deadline occurs. Therefore, it only forwards the query responses for phone

applications for "The Stinking Rose" and "Il Fornaio."

The `MC` receives the query responses and extracts the name from the `APprof` and displays them to the user. The user has eaten at "Il Fornaio," and wanting to try something new, chooses "The Stinking Rose." The `MC` sends the `APID` to the query server, which relays it to application server #2. Application server #2 consults its application database and sends the application to the query server. Again the query server consults its schedule, and determines that the application will arrive in time to meet the deadline. It forwards the application to the `MC`. The `MC` receives the application and executes it.

3 Spatial Queries

The query submitted by a mobile client to a query server contains both location-based constraints and non-location-based constraints [9]. For example, in the query "What is the closest Italian restaurant?", "P(x): x is an Italian restaurant" is a non-location-based constraint, whereas "Q(x): x is the closest to the mobile client" belongs to location-based constraints. Along with the query, the mobile client sends the deadline for the reply to the query, and its location.

A query server is responsible for several attachment points or cells. Therefore, a query sever is basically responsible for a local area, maintaining a spatial database that lists all the organizations that are interested in the local area. Specifically, for each organization, the database lists the part of the local area in which the organization is interested, and the application server hosting its applications. The interest area can be represented by a polygon, and the application server can be represented by its URL address. After receiving the query from a mobile client, the query server searches its spatial database to determine the scope of the search. Specifically, the query server determines which application servers are involved in the search based on the location of the mobile client. Based on the deadline from the mobile client, the query server establishes a new reply deadline for the application servers involved in the search. This new deadline is also soft. The query server decides it based upon experience, that is, the query server collect statistics, and uses the statistics as the basis for deadline assignment.

The main research issue involved here is how to answer a location-dependent query constrained by real-time deadlines. Although a significant amount of research has been reported in the literature, no one has yet addressed the real-time issue in processing spatial data queries[10,11]. We first define exactly what kind of spatial data and queries the system will support. To do this, we assume there are two kinds of mobile clients based on their moving

speeds; one is pedestrian and the other is vehicular. The spatial database we define consists of three layers. The first layer describes the paths that can be used by a pedestrian. The second layer specifies the roads that can be used by a car. Finally, the third layer defines the positions of all participating organizations. Organizations can be represented by point data or region data. The basic queries involving the spatial data include spatial range queries, nearest neighbor queries, and join queries [12–14].

This research issue can take two distinct approaches. One is to use an existing commercial DBMS with a spatial data option to build a system which can handle real-time constraints, as described in Section 3.1. The other is to have a spatial database that has real-time constraints integrated into it, as presented in Section 3.2.

3.1 Spatial Database with a Real-time Facade

Although some database management systems, such as the Oracle DBMS, have included a spatial data option, no one has addressed how to meet the deadline when querying spatial databases [15]. We can place a facade around an existing spatial database to achieve a certain amount of real-time capability. In this approach, a wrapper can be added at the application level that is able to schedule the execution of queries, and to collect the statistics needed to build the knowledge base that will be used to schedule the execution of later queries. This knowledge base will store the statistics distinguished by the different kinds of queries, as described above. The goal here is to maximize the number of queries answered within their corresponding deadlines.

3.2 Integrated Real-time Spatial Database

The ultimate goal is to build a DBMS that handles both spatial data and real-time constraints. To meet this goal, it may be necessary to modify existing practices in how to represent spatial objects and indexing spatial data to meet real-time constraints for query execution. Such a spatial database has four kinds of basic objects: vertex, edge, loop, and face. It is assumed that a complex object consists of a list of faces. Each face consists of one or more loops, one of which is external while the remainder are internal. Each loop is a list of ordered edges or vertices. Recently, old geometric algorithms have been used to solve problems in spatial databases [16,17]. One such approach, is the winged edge structure [18]. It has been used to successfully represent 3-D geometric models [19–21]. In this structure, each edge points to two vertices, and to the two loops to which it belongs: left and right. Each edge also lists its previous and next edges in its left and right loops. A vertex points to any edge that has the

vertex as its endpoint, and a loop points to any edge in the loop. In addition to the topological information represented by the winged edge structure, we can add geometric information to vertices and edges. For example, vertices can be associated with coordinates. There are at least two major advantages in using the winged edge structure. First, the topological structure will not change if the associated geometric information changes within certain limit. Second, all information about an object, e. g. all edges in a loop, is readily available.

With the winged edge structure, it is easy to answer the three kinds of basic spatial data queries: spatial range queries, nearest neighbor queries, and join queries. To answer a spatial range query, the vertices that fall within the range can be easily found. From the winged edge structure, we find the related edges, loops, and faces from those vertices. To answer a nearest neighbor query, a the nearest vertex is to be found first, and the vertex can be used to obtain the object. To answer a join query, the vertex, edge, or face set can be used to reduce the search time. Research needs to be done on how to create indexes on objects or their constituent elements such that the number of page IOs related to each kind of queries is upper-bounded, preferably minimized. The upper-bound of the number of page IOs will greatly help scheduling of the execution of spatial data queries in a real-time environment.

Since any face can be triangulated [22], a special case of the winged structure can be considered. In this special case, a face has only one loop with three edges. Therefore no loop object is needed. A three edged face, or triangle, has many properties such as simplicity and convexity. The performance differences between the special and general cases needs to be studied.

4 Transaction Scheduling

In this mobile distributed application delivery system, scheduling strategies are utilized by the query server, application server and client. The client must schedule tasks from the network as well as local tasks such as an alarm for appointments. The query server must schedule client requests that it sends to the application servers, and it also schedules the results to be sent to the clients. The query server receives requests from mobile clients for information, with most of the requests being read requests. However, our strategies also accommodate update requests to a database, such as a client wishing to make a reservation at a restaurant. An application server must schedule access to the database and manage its concurrency control. As a result, the requests in our application delivery system are considered to be transactions that have timing constraints and require concurrency control to maintain the consistency of the database. The scheduling requirements for such real-time transactions needs

to be studied.

The scheduling of transactions has been an important area of research in real-time systems. While a conservative estimate of the time it will take for a transaction to execute can be made, in general, interference that will occur among the transactions cannot be predicted. Concurrency control for real-time transactions [23–26] combines time critical scheduling as well as the properties of traditional concurrency control algorithms. The concurrency control techniques are based on 2-phase locking or optimistic concurrency control. Scheduling decisions can be made based on deadline, such as scheduling the transaction with the earliest deadline first (EDF) or scheduling the transaction with the least slack time (the amount of time remaining before the deadline after execution time is taken into account).

4.1 Deadline Classification

The deadlines in a real-time system can be soft, firm, or hard deadlines [27]. Conventional transactions that have response time requirements can be thought of as soft real-time transactions [28]. We assume that the majority of deadlines for our delivery system are soft deadlines. A soft deadline may be missed, yet the result produced still has some value that monotonically decreases with time after the deadline. As stated earlier, users requesting information will typically not be willing to wait for an unlimited amount of time before receiving a reply to a query.

There will also be requests that have firm deadlines in this system. While failure to produce a result by the deadline is not catastrophic for a firm deadline, the results produced after the deadline are useless. An example of this is a client requesting information related to exits on the interstate, while driving past such exits. Once the driver has passed an exit, any information related to the exit is useless. Lastly, it is also possible for our system to have hard deadlines, where failure to respond to such a request by the deadline can have catastrophic consequences. An example is a client requesting information about nearby hospitals because of a medical emergency. It is therefore, important for this delivery system to accommodate different types of deadlines.

4.2 Modeling Transactions

We assume each transaction T_i can be modeled as: $(A_i, D_i, E_i, P_i, R_i, C_i)$, where A_i is the arrival time of the task, D_i the deadline, E_i the execution time, P_i the priority of the transactions, R_i indicates the request type (read

or update), and C_i describes the machine platform capability (`MCprof`). While we assume that a transaction in our system may be an update, we assume that the percentage of update transactions in our system will be very small. Transactions will be assigned priorities based on criticality and deadline. The priority can change dynamically as the task changes. For example, as a mobile user drives closer to an interstate exit, the criticality of the location dependent query increases. Scheduling the requests requires knowledge of the execution time of the task and the system will provide a means of computing the execution time E_i of each task based on the request type. Since the mobile computing devices of each client will have different capabilities, E_i can also be based on the platform on which a result will execute, when scheduling for the query server.

4.3 Approximation for QoS

One possible way to meet deadlines is to provide multiple levels of quality of service to the clients. The different levels of quality of service can be viewed as providing an approximate answer or an imprecise answer to a query. The imprecise computation approach [29,30] has been used in hard real-time scheduling as a means to provide an answer to a real-time task when an exact answer cannot be provided by the deadline. The strategy is to divide each task into a mandatory and an optional part. As long as each task is able to execute its mandatory part by the deadline, no deadline is missed. Any additional processor time is assigned the optional part. Approximate query processing [31,32] is a strategy to provide approximate answers to real-time database queries. Approximate query processing is based on the imprecise computation approach, which trades off the quality of the result for the time to produce the result. The approximate answers converge to the exact answer, so if there is enough time to retrieve and process all of the data, the exact answer is provided. Otherwise, the best approximation produced is provided as the approximate answer by the deadline [33–39].

In the delivery system providing approximate answers to queries is considered. An exact answer E to a query in the delivery system consists of one or more items, with one or more version of each item that varies in quality, such as visual quality. Approximate answer A consists of one of more of the items in the exact answer E and can vary in the number of items, as well as the visual quality of each item, such that $A \subseteq E$. The approximate answers converge to the exact answer if there is enough time to retrieve and process all of the data. A strategy needs to be developed to measure the quality of an approximate answer based on the number of data items returned, the processing time required and the visual quality of the result.

The system does not provide a guarantee to meet deadlines, but instead attempts to minimize the number of deadlines missed and maximize the quality of the answers. It is assumed that requests to the delivery system are aperiodic and will not be known beforehand. The static information, such as the deadlines and performance profile of the task, are used to make scheduling decisions. However, an adaptive policy [40,41] that considers dynamic information, such as system load [28] can be considered. The quality of service provided not only depends on the load of the system, but can also depend on the capabilities of the client machines that receive the information. For example, a user requesting restaurant information can receive an animation or a simple HTML page providing knowledge of the food offered at the restaurant, depending on the capability of the client.

Adaptive scheduling policies that consider dynamic information, such as system load and the quality of the answers provided, for real-time transactions should be developed and studied. Approximate answers are provided as one strategy for satisfying the deadlines of the transactions. Strategies for measuring the quality of approximate answers returned to the client are also developed. These policies can be evaluated using simulation studies, which can determine which strategies to implement in a mobile delivery system.

5 Performance Profiles

In a system where only "dumb data" is being delivered, simple metrics, such as the size of the data, is sufficient to determine where information should be processed. This is a well-known technique in distributed databases, where decisions regarding the processing of a query are based upon data size estimates[42]. In this system, to effectively schedule applications, such a simple approach will not suffice. In the most basic case, there must be an estimate made of how long an application will take to execute on the intended mobile client. The information needed to make this determination includes a machine independent profile for both the application and the mobile client that can be used to calculate the amount of time needed to execute the application. For the application, this information is gathered off-line and stored along with the application in its profile on the application server. The first approximation of a machine-independent application profile would be a simple count of the number of CPU clock cycles needed to run the application to completion. Many such tools exist for gathering information about Java programs [43]. Jones has developed an annotation-aware Java Virtual Machine which does machine-code generation based upon optimization summaries added off-line[44]. This system could be modified to gather performance information and calculate performance estimates.

The mobile client calculates an estimate of the amount of time required to execute the application by multiplying the application's clock cycle count by the clock rate of the mobile client's processor. This provides a crude estimate of the execution time. The execution time estimate matches exactly only if the mobile client exactly matches the environment where the application was profiled (i. e. same processor, same clock rate, same resident libraries.)

5.1 Enhanced Performance Model

Although exact estimates of execution time are not needed to do soft real-time scheduling, the more accurate the estimates, the more applications that can be scheduled and executed to completion. Therefore, experiments should be done with a series of improvements to the time estimates. The first factor to experiment with is RISC versus CISC. A reduced instruction set computer (RISC) typically accomplishes more work per cycle, due to a lower cycles per instruction rate, than complex instruction set computers (CISC). Moving away from measures of processor performance, other aspects of an applications resource requirements should be examined. Mobile computing devices vary greatly in their graphic display capabilities, ranging from 3×15 character cellphones to monochrome 150×200 pixel devices in low-end PDAs to 24-bit color 1600×1240 laptop displays. The decision of whether an application can be run on a specific device is not as simple as deciding the largest/deepest image used by the application, however. A device with a less capable display can remediate by reducing the bit-depth and by scaling. However, this may require additional processing time, which must be recognized when determining how long the application takes to execute.

The previous performance measures have only dealt with time to completion. In a networked environment, the application may make dynamic demands for information from an application server or some other information source. If such information requests are periodic, such as streaming audio or video, then the mobile client's networking hardware must have the bandwidth to handle these requests. The mobile client must also account for current network congestion when determining if an application can be scheduled.

5.2 Performance Profile Consumers

Where is this performance profile information used? It is used by the query server as a filter to determine which applications should be presented to the user for choosing. Placing this decision process on the query server reduces the amount of information transmitted on the wireless network between the mobile client and the attachment point, the probable bottleneck of our system.

This information is also used by the mobile client to dynamically schedule applications. For example, if the network is currently congested, then applications which do not use the network will be given preference over those that do.

5.3 Application Tailoring

As mentioned above, applications can be tailored by altering graphics data by "de-resing." Another technique is to tailor the program for the specific client. This can be done by following the typical design technique of separating presentation from domain models to an even greater extent than is currently done. Current techniques for dealing with platform independence usually rely on creating the GUI using some sort of Abstract Factory [45] to create widgets for the specific environment. The underlying assumption is that any environment that the program will run on will have roughly the same amount of screen real-estate and further, that it will have a graphic interface. There is also the assumption that all platforms will have adequate memory to hold the code for a rich GUI. This is not an adequate approach for a mobile application environment.

The *application stitcher* is the component of the application server responsible for tailoring applications for their environment. To do this tailoring (*stitching*), it uses the *performance profile*. This information is used to select appropriate applications and to produce an application tailored for the requesting device. The selection of applications is done based upon the application's performance requirements. These requirements are stored as annotations to the `.class` files, in a fashion similar to the "JIT-hints" used by annotation-aware JVMs [44,46,47]. The next step is for the application server to produce an application tailored for the requesting device. Program size can be manipulated to affect a tradeoff between ease of use and resource consumption. Decreasing program size decreases the amount of time to transfer the application and the amount of memory used by the device. The *stitcher* can perform several transformations to the most elaborate version of the program to reduce its memory size.

In addition to the image resolution reduction technique, another technique—one that relies on a constrained program structure—is to omit edit checking and guides from the user-interface component of the application. An example transformation might be to change a GUI widget from a "select from a list" to a simple text entry box. Of the possible transformations, such as "de-resing" images, eliding edit checks, etc, the most difficult is for the *stitcher* to adjust to the interface of the device. Superficially, there seems to be little similarity between voice-mail systems with speech recognition and large display laptops running a GUI-based application. However, all user-driven programs have an

underlying core concept—streams of events and the associated response to them. The events are generated by user actions such as saying "Baltimore" in response to the prompt "Destination city?" or by selecting "Baltimore" from a drop-down box. We have identified two distinct user-interface modes that exhibit the stream of events model. The first is the query-response model typically used for telephone systems or speech driven applications. The second is the graphical user-interface model (GUI). The two are differentiated by the number of possible events the user can generate at a given instant. A user of a GUI typically has many choices of events to generate, such as pressing buttons, selecting from menus, moving scroll-bars, etc. Command-based speech applications, as opposed to dictation systems, typically have a limited set of spoken commands that are available at any point in time. Our system has two special-purpose programming languages for specifying the user-interface, and uses value-objects to communicate with the application's domain model.

5.4 Query-Response Language

The query-response or conversational model accommodates a wide variety of human-computer interaction modalities. In Figure 3 we see a program for specifying the user-interface for querying a book database. The language uses a state machine to model the flow of events and actions. The domain objects are made available through the import statement. The verbs, such as `say`, are translated into calls into a *context* object. Different context objects provide services appropriate to the environment—speech, text strings, etc. The *stitcher* chooses the appropriate *context* object based upon the performance profile.

5.5 GUI Language

One approach to fitting user interfaces to small displays is one taken by web browsers for PDAs—scrolling. This works, but applications tailored for small displays are much more user-friendly. Moreover, as display sizes shrink, programs written for "large" screens stop being viewable. Our GUI language provides a mechanism for specifying that collections of widgets belong together because of their application semantics similarity. In particular, we use the ValueModel[48] pattern to cleanly separate the application model from the GUI. Further, the language specifies a set of widgets that are abstract enough that they can be translated into a broad variety of concrete GUI toolkits. In the Java arena, large display devices use the Swing framework, while handhelds use the javax.microedition.lcdui, which is much more limited. Our GUI language, however, is not intended to allow for only a lowest common denominator interface. Instead, it will shed features at application tailoring time to

```
import java.io.FileReader;
import search.Result;
...
BEGIN {
StringSearcher searcher = new StringSearcher();
searcher.load(new FileReader("test.dat"));
}

searchPrompt:
    say "Please " inputVerbVocative
        " an Author, Book, or Year";

searchValue:
    searchValue = textEntry();

doSearch:
    if (confirmAction("You " inputVerbPastTense
                      searchValue "."
      "Is this correct?")) {
        results = doQuery(searchValue);
        continue;
    } else
        goto SearchPrompt;

searchResultsDisplay:
    if (results.size() == 0)
        say "No matching books found";
    else
        resultsLoop(results);

function resultsLoop(Result results) {
    say results.size() " books found";
    repeat with document in results {
        say "Author " (author of document)
        say " Book "  (book of document)
        if (!confirmAction("Continue? ")) end repeat;
    }
}

function doQuery(String q) returns Result {
    return searcher.find(new Query(q));
}
```

Fig. 3. Example Query-response program

match the target device. The feature shedding is guided by the programmer's explicit marking of features which may be omitted.

5.6 Profiling Research Goals

What additions to existing technology are needed to support performance profiling? The first step is the off-line profiling of applications. Since the most pervasive mobile code language is Java, existing tools for gathering processor performance information can be used [49]. This work needs to be augmented to produce information about images and streaming network usage. One possible way of obtaining this additional information is to take advantage of the machine-independent nature of Java and augment existing class libraries to record additional profile information. These modified class libraries can then be used in any mobile computing device supporting Java.

The second step is the dynamic use of profile information by the mobile client. Here, the JVM's thread scheduler must be modified to accommodate real-time scheduling and the use of the dynamic information about network congestion and network resource use.

6 Scheduling Wireless Transmissions

Any object to be delivered to the mobile client (MC) will have to go through an attachment point (AP). APs are mobile base stations each serving a geographical region. Typically wireless communication links are high error-rate, low-bandwidth links and therefore easily saturated. In addition, any transmission on such links is slotted, i.e., time is divided in frames and a frame is divided into slots. To transmit data, one needs to reserve one or more slots in one or more frames depending on the size and priority of the data. Further, as the load will keep changing, to satisfy the real-time constraints, some scheduling and other optimization at the network modules (in transport layer and in DLC layer) without further intervention of the query server and other database entities helps to adapt to varying loads not associated with application delivery.

It is to be noted that the applications at hand are real-time applications. Hence, in general, it is important to ensure that for hard real-time applications all dead lines are met and for soft real-time applications the penalty function for missing deadlines are minimized. In particular, the following optimization problems need to be addressed.

(1) **Scheduling hard real-time traffic within a cell.** Hard real-time traffic receives higher priority over soft real-time traffic. So far as the network is concerned, a hard real-time application may be modeled as a set of messages $M_i = (L_i, A_i, D_i, T_i), i = 1, \ldots, n$, where L_i is the length of the message in bits, A_i is the arrival time (may be a function of A_{i-1}), D_i is the deadline and T_i is the message direction (AP to MC or MC to AP). Given a set of such applications, it is necessary to use slot reservation strategies (in the MAC layer) such that the deadlines of all hard real-time applications are satisfied (if possible) and maximum amount of bandwidth becomes available for soft real-time applications. Solutions to the problem is known when the messages within an application are of equal length, periodic, unidirectional and $A_i = D_{i-1}\ \forall i$ [50]. However, such restricted methods do not solve the more general problem.

(2) **Scheduling soft real-time traffic within a cell.** The soft real-time traffic also involves similar scheduling problems as hard real-time traffic. However, for soft real-time applications a penalty function is required that is an increasing function of the tardiness of the messages. This penalty function captures the probability that a user will find the system slow enough to quit the system. While a linear function of tardiness is both conceptually and algorithmically attractive, it is not appropriate for this system. Generally there is a limit beyond which the penalty function should grow quickly. A linear function does not have this property. An appropriate penalty functions must be devised and schedules computed to minimize the aggregate penalty. For soft real-time traffic scheduling in a slotted environment, please see [51]

(3) **Choice between alternative versions.** As described earlier, the documents requested by the users may have multiple versions of varying size with varying visual appeal and varying processing load. A user may quit if the visual appeal decreases and/or the processing load increases. It is reasonable to assume that the size of a document increases with visual appeal and decreases with the processing load to the MC (otherwise some of the alternate versions are not meaningful choices). There is a penalty function attached to the quality of a presentation which is a function of the visual appeals and processing loads of all messages in that presentation. Clearly, the network load and hence the tardiness penalty increases with the size of documents while the quality penalty goes in opposite direction. This observation presents the interesting problem of choosing appropriate versions dynamically (or even off-line) to minimize the aggregate tardiness and quality penalty. As the network load may considerably vary unexpectedly in short notice (for example if some packet in an hard-real time application needs to be retransmitted within a short time interval), it is a good idea to cache a few different versions in the AP and allow the transport layer to pick the appropriate version at the time of transmission. In addition to the choice algorithm, this requires a non-traditional *application program interface*. The sliding window type

protocol to be used will also have to be changed. The sliding window must be modified so the unit of transmission will be an application/document instead of bytes or packets—further, the sender will need feedback about how much of the application has been lost in transmission. This information is necessary to determine whether to retransmit the old version or try another low-cost/high-cost version. To the best of our knowledge, this is a novel approach which has not been tried previously.

Admission control is another open area. If the system has some idea about the offered load by a new request and it finds that the new connection may lead to higher aggregate penalty, then the new request should be denied admission.

(4) **Consistent priority assignment to objects.** It is conceivable that some users will find some objects more important than other. For example, a restaurant advertiser will find the picture of a bowl of steaming soup more important than the picture of a flower arrangement even though he will probably like both of them. If the system provides for multicasting, then the system may decrease some load by multicasting objects (instead of unicasting to each user) important to many users. However, the objects should be multicast in the order of their importance or priority. The transport layer will then transport as many objects as early as possible (given network load), beginning with the highest priority object and delaying (or choosing to drop) the remaining. This again requires a change in the API as well as in the sliding window protocol. In addition, a priority assignment algorithm must be devised. This will require a novel organization of the sliding window buffer. Instead of a linear ordering of the buffer, we shall need a tree-like ordering to accommodate the priority ordering of documents. For a survey of multicasting mechanisms as well as effect of data loss please see [52].

The following example will show that assigning consistent priority is not a trivial task. Let there be three objects a, b and c. Let three users arrange them (in the order of decreasing priority) as a, b, c; b, c, a and c, a, b. Here a should get more priority over b as two users find it more important, and b should get higher priority over c for the same reason. But if we do that, a will get higher priority over c which we can not do as two users find c more important than a.

Allowing the users to assign a priority number is not a good solution as the range of the priority numbers will always be questionable and a user may not even know exactly by how much they prefers one object over other. What we propose here is to algorithmically investigate priority schemes that violate the fewest priority constraints and/or affects the least number of users.

(5) **Handoff** As an MC moves from one AP to another, the new AP will have to provide necessary services and will be responsible to keep the penalty low. To facilitate this, mobility models must be developed to warn an AP about expected changes in future load. This information will help an AP

not to over commit itself in terms of bandwidth while not unnecessarily denying many admissions. It is to be noted that only the DLC layer of an MC is aware of an imminent handoff. One solution is to modify the MAC layer protocol to allow it to inform the current AP about an imminent handoff so that the current AP may inform the prospective new AP about the possible new MC entering its zone. However, the exact time of the handoff is a statistically variable entity which may not be accurately predicted. The prospective new AP needs to reserve resources using some prediction algorithm such that bandwidth is not unnecessarily wasted or too many connections are not denied admission due to the expected handoff. Appropriate MAC protocols and prediction algorithms need to be investigated.

7 Conclusions

We have outlined an architecture for achieving tighter integration of application and information distribution that is spatially sensitive. The first step was defining a division of labor between system components that more closely follows the way that infrastructure, information, and application services are currently deployed. Next, we showed how simple queries could result in delivery of applications that are tailored for the location and capabilities of the mobile client. We then defined the details of how the existing underlying communications, application serving, and database infrastructure must be modified to accommodate soft real-time constraints. This architecture provides a guideline for the next step in mobile, ubiquitous computing—the provisioning of applications and information that is highly adapted to its environment.

References

[1] T. Lindholm, F. Yellin, The Java Virtual Machine Specification, The Java Series, Addison-Wesley, 1997.

[2] Hewlett Packard, Inc., coffee and a hotspot, http://www.cooltown.com/mpulse/0902-starbucks.asp (September 2002).

[3] Handspring, Inc., Treo 180 overview, http://www.handspring.com/products/treo/index.jhtml.

[4] J. Cai, D. J. Goodman, General packet radio service in GSM, IEEE Communications Magazine 35 (10) (1997) 122–131.

[5] Sun Microsystems, Inc., JNDI: Java^{TM}naming and directory interface^{TM}, ftp://ftp.javasoft.com/docs/j2se1.3/jndiexecsumm.pdf.

[6] Sun Microsystems, Inc., Jini^{TM}technology architectural overview, http://wwws.sun.com/software/jini/whitepapers/architecture.html (January 1999).

[7] L. Gong, Project JXTA: A technology overview, http://www.jxta.org/project/www/docs/jxtaview_01nov02.pdf (November 2002).

[8] E. Guttman, Autoconfiguration for ip networking: Enabling local communication, IEEE Internet Ccomputing (2001) 81–86Http://www.zeroconf.org/w3onwire-zeroconf.pdf.

[9] T. Imielinski, B. R. Badrinath, Querying in highly mobile distributed environments, Proc. 18th Int'l Conf Very Large Data Bases .

[10] D. Barbara, Mobile computing and databases - a survey, IEEE Transactions on Knowledge and Data Engineering 11 (1) (1999) 108–117.

[11] S. Shekhar, S. Chawla, S. Ravada, A. Fetterer, X. Liu, C. Lu, Spatial databases - accomplishments and research needs, IEEE Transactions on Knowledge and Data Engineering 11 (1) (1999) 45–55.

[12] R. Ramakrishnan, J. Gehrke, Database Management Systems, McGraw-Hill, 2000.

[13] Y. Theodoridis, E. Stefanakis, T. Sellis, Efficient cost models for spatial queries using R-trees, IEEE Transactions on Knowledge and Data Engineering 12 (1) (2000) 19–32.

[14] G. Proietti, C. Faloutsos, Analysis of range queries and self-spatial join queries on real region datasets stored using a R-tree, IEEE Transactions on Knowledge and Data Engineering 12 (5) (2000) 751–762.

[15] O. Corporation, Oracle8i spatial user's guide and reference, http://technet.oracle.com/doc/inter.815/a67295/toc.htm.

[16] S. Berchtold, D. A. Keim, H. P. Kriegel, T. Seidl, Indexing the solution space: A new technique for nearest neighbor search in high-dimensional space, IEEE Transactions on Knowledge and Data Engineering 12 (1) (2000) 45–57.

[17] A. Corral, Y. Manolopoulos, Y. Theodoridis, M. Vassilakopoulos, Closest pair queries in spatial databases, ACM SIGMOD on Management of Data (2000) 189–200.

[18] B. G. Baumgart, Geometric modeling for computer vision, Ph.D. thesis, Stanford University (1974).

[19] M. Mantyla, A. Sulonen, Gwb: A solid modeler with euler operators, IEEE Computer Graphics and Applications 2 (7) (1982) 17–31.

[20] H. Chiyokura, F. Kimura, A method of representing the solid design process, IEEE Computer Graphics and Applications 5 (4) (1985) 32–41.

[21] J. Zhang, Torus: A basic-operator based solid modeling system, Master's thesis, Zhejiang University (1987).

[22] F. P. Preparata, M. I. Shamos, Computational Geometry: An Introduction, Spring-Verlag, 1985.

[23] S. Abbott, H. Garcia-Molina, Scheduling real-time transactions: A performance evaluation, ACM Transactions on Database Systems 17 (3) (1992) 513–550.

[24] A. Buchmann, D. R. McCarthy, M. Hsu, Time-critical database scheduline: A framework for integrating real-time scheduling and concurrency control, Proceedings 5th International Conference on Data Engineering (1989) 470–480.

[25] J. R. Haritsa, M. Livny, M. J. Carey, Earliest deadline scheduling for real-time database systems, Proceedings of Real-Time Systems Symposium (1991) 232–242.

[26] L. Sha, Concurrency control for distributed real-time databases, SIGMOD Record 17 (1) (1988) 82–98.

[27] V. F. Wolfe, L. C. DePippo, Real-time database systems, in: P. J. Fortier (Ed.), Database Systems Handbook, McGraw Hill Publishers, 1997.

[28] S. Chakravarthy, D. Hong, T. Johnson, Incorporating load factor into the scheduling of soft real-time transactions, Tech. rep., University of Florida (1994).

[29] J. Y. Chung, J. W. S. Liu, K. J. Lin, Scheduling periodic jobs that allow imprecise results, IEEE Transactions on Computers 39 (9) (1990) 1156–1174.

[30] K. J. Lin, S. Natarajan, J. W. S. Liu, Imprecise results: Utilizing computations in real-time systems, Proceedings IEEE Real-Time Systems Symposium (1987) 210–217.

[31] S. V. Vrbsky, A data model for approximate query processing of real-time databases, Data and Knowledge Engineering 21 (1) (1996) 79–102.

[32] S. V. Vrbsky, J. W. S. Liu, Approximate: A query processor that produces monotonically improving approximate answers, IEEE Trans. on Knowledge and Data Engineering 5 (6) (1993) 1056–1068.

[33] N. Jukic, S. V. Vrbsky, Temporal aggregates for time-constrained queries, Proceedings of IASTED Intelligent Information Management Systems (1996) 21–25.

[34] N. Jukic, S. V. Vrbsky, Approximate aggregates, Information Systems 21 (7) (1996) 595–614.

[35] S. V. Vrbsky, S. Tomic, N. Jukic, Concurrency control for approximate query processing of real-time databases, in: A. Bestavros, V. Fay-Wolfe (Eds.), Real-Time Database and Information Systems: Research Advances, Kluwer Academic Publishers, 1997, pp. 227–246.

[36] S. V. Vrbsky, N. Jukic, Approximate query processing of temporal data in real-time databases, Proceedings of ISMM Intelligent Information Management Systems (1994) 21–25.

[37] S. V. Vrbsky, N. Jukic, Analysis of approximate query processing overhead, Proceedings of ISMM Intelligent Information Management Systems (1995) 21–25.

[38] S. V. Vrbsky, S. Tomic, Satisfying timing constraints of real-time databases, Journal of Systems and Software 41 (1) (1998) 63–73.

[39] S. V. Vrbsky, S. Tomic, Satisfying temporal consistency constraints of real-time databases, Journal of Systems and Software 45 (1) (1999) 45–60.

[40] S. M. Tseng, Y. Chin, W.-P. Yang, An adaptive value-based scheduling policy for multiprocessor real-time database systems, Proceeding 8th International Workshop on Database and Expert Systems (1997) 254–259.

[41] H. R. Chen, Y. Chin, S. M. Tseng, Scheduling value-based transactions in distributed real-time database systems, Proceeding 15th International Parallel and Distributed Processing Symposium (2001) 978–984.

[42] R. Elmasri, S. Navathe, Fundamental of Database Systems, 3rd Edition, Addison-Wesley, 2000.

[43] T. Newhall, B. P. Miller, Performance measurement of dynamically compiled Java executions, Proceedings of the ACM Java Grande Conference (1999) 42–50.

[44] J. Jones, S. Kamin, Annotating Java class files with virtual registers for performance, Concurrency: Practice and Experience 12 (6) (2000) 389–406.
URL
`http://www3.interscience.wiley.com/cgi-bin/abstract/72515727/START`

[45] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns, Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1994, iSBN 0-201-63361-2.

[46] A. Azevedo, A. Nicolau, J. Hummel, Java annotation-aware just-in-time (AJIT) compilation system, in: ACM 1999 Java Grande Conference, 1999.

[47] C. Krintz, B. Calder, Using annotations to reduce dynamic optimization time, in: Proceedings of the ACM SIGPLAN '01 Conference on Programming Language Design and Implmenetation (PLDI), ACM Press, 2001, pp. 156–167.

[48] B. Woolf, Understanding and using the valuemodel framework in visualworks smalltalk, in: J. O. Coplien, D. C. Schmidt (Eds.), Pattern Languages of Program Design, Addison-Wesley, 1995, pp. 466–494.

[49] T. Newhall, B. P. Miller, Performance measurement of dynamically compiled Java executions, Concurrency: Practice and Experience 12 (6).

[50] C.-C. Han, K. G. Shin, Message transmission with timing constraints in ring networks, Proc. of 17th IEEE Real-Time Systems Symposium (1996) 165–174.

[51] R. Kannan, S. Ray, R. Bartos, An improved optical switch for group communication, International Conference on Advanced Computing and Communications (ADCOM) (2000) 205–211.

[52] R. Kannan, S. Ray, A modular scalable multicast atm packet switch with low delay and hardware complexity, IEEE/ACM Transactions on Networking 8 (3) (2000) 407–418.