A

Seminar report

On

Rover Technology

Submitted in partial fulfillment of the requirement for the award of degree
Of CSE

SUBMITTED TO:

www.studymafia.org

SUBMITTED BY:

www.studymafia.org

www.studymafia.org

Acknowledgement

I would like to thank respected Mr…….. and Mr. ……..for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a seminar report. It helped me a lot to realize of what we study for.

Secondly, I would like to thank my parents who patiently helped me as i went through my work and helped to modify and eliminate some of the irrelevant or un-necessary stuffs.

Thirdly, I would like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Next, I would thank Microsoft for developing such a wonderful tool like MS Word. It helped my work a lot to remain error-free.

Last but clearly not the least, I would thank The Almighty for giving me strength to complete my report on time.

www.studymafia.org

Preface

I have made this report file on the topic **Rover Technology**; I have tried my best to elucidate all the relevant detail to the topic to be included in the report. While in the beginning I have tried to give a general view about this topic.

My efforts and wholehearted co-corporation of each and everyone has ended on a successful note. I express my sincere gratitude to …………..who assisting me throughout the preparation of this topic. I thank him for providing me the reinforcement, confidence and most importantly the track for the topic whenever I needed it.

www.studymafia.org

CONTENTS

www.studymafia.org

INTRODUCTION

Location-aware computing involves the automatic tailoring of information and services based on the current location of the user. We have designed and implemented Rover, a system that enables location-based services, as well as the traditional time-aware, user-aware and device-aware services. To achieve system scalability to very large client sets, Rover servers are implemented in an "action-based" concurrent software architecture that enables fine-grained application-specific scheduling of tasks. We have demonstrated feasibility through implementations for both outdoor and indoor environments on multiple platforms.

A user is shopping in a mall. On entering a store, he pulls out a PDA and browses through detailed information about the products on display. Satisfied with the information, through the PDA, he makes an online purchase of the items of interest that will be subsequently shipped to his home directly. As he walks on to the next store, which happens to be a video rental store, information on newly-released movies in his favorite categories are downloaded automatically into his PDA, along with their availability information. He chooses a couple of these movies and indicates that he will pick them up at the storefront. His membership discounts are applied to the bill, and he confirms the charge to his credit card. The intriguing aspect of this scenario is the automatic tailoring of information and services based on the current location of the user. We refer to this paradigm as **location-aware computing**. The different technology components needed to realize location-aware computing are present today, powered by the increasing capabilities of mobile personal computing devices and the increasing deployment of wireless connectivity (IEEE
802.11 wireless LANs [7], Bluetooth [1], Infra-red [2], Cellular services, etc.) What has hindered its ubiquitous deployment is the lack of system-wide integration of these components in a manner that scales with large user populations. In this paper, we describe the design and initial implementation
experience of such a system, which we call *Rover*, and discuss the impact such a system can have on the next generation of applications, devices, and users.

Location-aware, in addition to the more traditional notions of time-aware, user-aware, and device-aware. Rover has a location service that can track the location of every user, either by automated location determination technology (for example, using signal strength or time

www.studymafia.org

difference) or by the user manually entering current location (for example, by clicking on a map).

Available via a variety of wireless access technologies (IEEE 802.11 wireless LANs, Bluetooth, Infrared, cellular services, etc.) and devices (laptop, PDA, cellular phone, etc.), and allows roaming between the different wireless and device types. Rover dynamically chooses between different wireless links and tailors application-level information based on the device and link layer technology.

Scales to a very large client population, for example, thousands of users. Rover achieves this through fine-resolution application-specific scheduling of resources at the servers and the network.

We will use a **museum tour application** as an example to illustrate different aspects of Rover. We consider group of users touring the museums in Washington D.C. At a Rover registration point in a museum, each user is issued a handheld device with audio and video capabilities, say an off-the-shelf PDA available in the market today. Alternatively, if a user possesses a personal device, he can register this device and thus gain access to Rover. The devices are traceable by the Rover system. So as a user moves through the museum, information on relevant artifacts on display are made available to the user's device in various convenient forms, for example, audio or video clips streamed to the device. Users can query the devices for building maps and optimal routes to objects of their interest. They can also reserve and purchase tickets for exhibitions and shows in the museum later in the day. The group leader can coordinate group activities by sending relevant group messages to the users. Once deployed, the system can be easily expanded to include many other different services to the users. The next section gives a description of the kinds of services that are available through Rover. The successive sections provide an overview of the Rover architecture and a description of a concurrent software architecture that has been used for system scalability. The following sections expand on particular aspects of Rover, including clients, servers, data management and multi-Rover systems. Then we describe our initial implementation experience and conclude with ongoing and future work.

www.studymafia.org

LOCATION AWARE COMPUTING COMES OF AGE

REVIEW

At the core of invisible computing is *context awareness,* the concept of sensing and reacting to dynamic environments and activities. Location is a crucial component of context, and much research in the past decade has focused on location-sensing technologies, location-aware application support, and location-based applications. With numerous factors driving deployment of sensing technologies, location-aware computing may soon become a part of everyday life.

LOCATION-SENSING TECHNOLOGIES

A central problem in location-aware computing is the determination of physical location. Researchers in academia and industry have created numerous location-sensing systems that differ with respect to accuracy, coverage, frequency of location updates, and cost of installation and maintenance.

Coarse-Grained Systems

For applications in open, outdoor areas, the Global Positioning System isa common choice. A GPS receiver estimates position by measuring satellite signals' time difference of arrival. Although GPS offers near-worldwide coverage, its performance degrades indoors and in high-rise urban areas, and receivers have a relatively long start-up time and high cost. In 1989, Roy Want, Andy Hopper, and others pioneered the study of indoor location sensing with their infrared based Active Badge system. This provides room-grained location using wall-mounted sensors that pick up an infrared ID broadcast by tags worn by the building's occupants. Many of the location-sensing systems developed since then are based on radio. By using base station visibility and signal strength, it is possible to locate Wi-Fi-enabled devices with accuracies from several meters to tens of meters. Bluetooth technology, which offers a shorter range than Wi-Fi, can give more accurate positioning, but at the expense of requiring more fixed base stations to provide coverage. Inexpensive radiofrequency identification tags can be used for location determination as well by placing RFID readers at doorways and other strategic points to detect the passage of people or objects. Location information can also be derived from other types of

www.studymafia.org

RF infrastructures including those for mobile phones and TV broadcasts. These can be deployed over a wide area with relative ease, in contrast to technologies such as RFID that have limited transmission range. With mobile phones, Cambridge Positioning Systems has demonstrated location accuracies of 20meters, while Rosum has achieved accuracies from 3 to 25 meters with digital TV signals.

Fine-Grained Systems

Many of the above systems are based on technologies that were not developed with location sensing in mind. Perhaps as a consequence they exhibit modest accuracy, generally measured in meters. However, at least three types of systems have been designed specifically to provide fine grained location sensing, achieving accuracies on the order of centimeters. Ultrasound can be used to determine distances between mobile tags and known points in the environment. A process akin to triangulation can thebe employed to derive a location estimate for the tag. One type of ultrasonic ranging device is the Cricket indoor location system developed at MIT ,which is set to become available for purchase from Crossbow this year. Some computer vision-based systems are appealing because they do not require users to wear any sort of tag. However, such systems have difficulty identifying and simultaneously tracking many subjects. Vision-based systems using barcode-like tags tend to be more robust.. Ubisense, a company that builds real-time local positioning systems, recently demonstrated a fine-grained tracking system that uses ultra wide band radio signals. Unlike conventional radio signals, these signals can have pulse durations short enough to allow accurate time-of-arrival and angle-of-arrival measurement, with an accuracy of about 15 centimeters. In addition, ultra wideband technology does not require a direct line of sight between tags and sensors.

www.studymafia.org

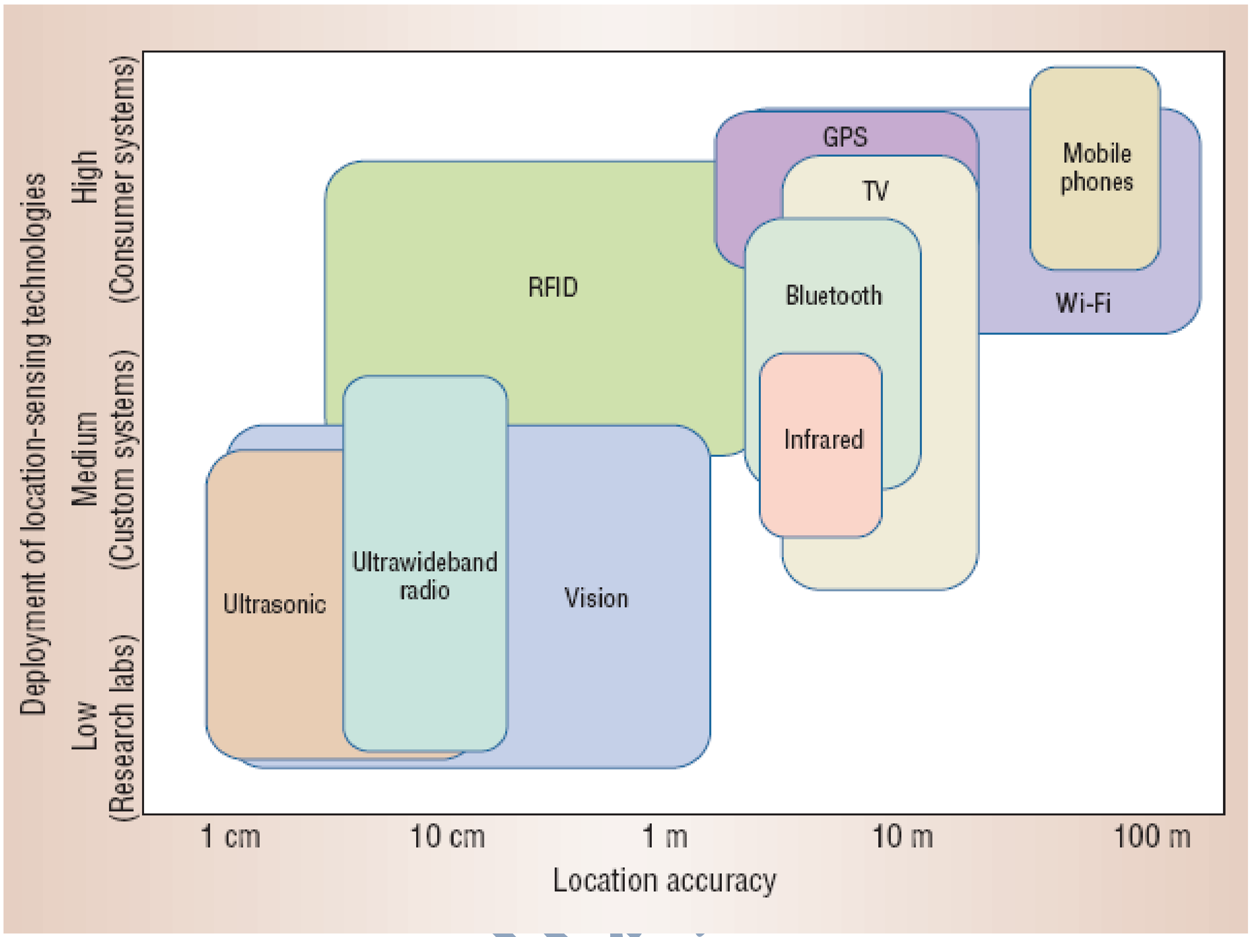


Fig 2.1 Location-sensing technologies

DEPLOYMENT

Figure 1 shows Each box's horizontal span shows the range of accuracies the technology covers; the bottom boundary represents current deployment, while the top boundary shows predicted deployment over the next several years and the current and predicted deployment of location-sensing technologies within the next two to three years. The widest existing deployments are based on GPS, which is particularly suited for outdoor applications. These include servicing applications centered on vehicle location such as route planning and fleet tracking, as well as applications integrated into handheld GPS units. Other current deployments are found in vertically integrated solutions and comprise a specific location-aware application, appropriate location-sensitive ing hardware, and a custom software platform. A handful of firms offer these systems in targeted application areas such as military training, human-body motion capture, supply chain management, and asset tracking. Looking ahead, numerous factor are accelerating the adoption of coarse-grained location-sensing technologies. To begin with, the

www.stu

recent explosion of Wi-Fi, Bluetooth, and other wireless networking technologies has led to many end-user devices being equipped with RF hardware that can be used for location sensing. In addition, the Enhanced 911 requirement—which mandates that US wireless carriers provide location accuracies of 50 to 100 meters for emergency 911 calls by the end of 2005— is driving incorporation of location sensing systems into mobile phones using GPS, base-station triangulation methods, and a combination of these technologies known as Assisted GPS. Similar requirements exist in the European Union.

SENSOR FUSION

Vertically integrated location-aware systems typically use one type of sensor for a single application. However, in the near future, many kinds of location sensors may be available to a particular client system. The task of
making sense of this vast amount of sometimes contradictory information, known as *sensor fusion*, presents a major challenge. Borrowing from the field of robotics, location researchers have settled on Bayesian inferencing as the preferred method for processing data from disparate location sensors. Using Kalman filters, hidden Markov models, dynamic Bayes nets, and particle filters, they have developed principled methods of incorporating sensor uncertainty as well as limits on speed and travel paths. The result is a location measurement derived from multiple sensors and constraints that uses a probability distribution rather than a single value to describe the inherent uncertainty. For example, researchers at the University of Washington have demonstrated an indoor location-measuring system that
processes data from multiple sources, including infrared and ultrasonic sensors, using a particle filter. In addition, the system learns typical walking paths through the building to aid in location estimation.

www.studymafia.org

ROVER ARCHITECHTURE DESCRIPTION

ROVER SERVICES

Rover offers two kinds of services to its users. We refer to them as *basic data services* and *transactional services*.

1. *Basic data services:* Rover enables a basic set of data services in different media formats, including text, graphics, audio, and video. Users can subscribe to specific data components dynamically through the device user interface. Depending on the capabilities of the user's device, only a select subset of media formats may be available to the user. This data service primarily involves one-way interaction; depending on user subscriptions, appropriate data is served by the Rover system to the client devices.
2. *Transactional services:* These services have commit semantics that require coordination of state between the clients and the Rover servers. A typical example is e-commerce interactions.

Services that require *location manipulation* are a particularly important class of data services in Rover. Location is an important attribute of all objects in Rover. The technique used to estimate the location of an object (some techniques are described in the Appendix) significantly affects the granularity and accuracy of the location information. Therefore an object's location is identified by a tuple of *Value*, *Error*, and *Timestamp*. The error identifies the uncertainty in the measurement (value). The timestamp identifies when the measurement was completed. The accuracy of the location information is relevant to the *context* of its use. For example, an accuracy of _ meters is adequate to provide walking directions from the user's current location to another location about 500 meters away. However, this same accuracy is inadequate to identify the exhibit in front of the user. User input in these cases, helps significantly improve the accuracy of user location information.

Map-based services are an important component of location manipulation services. Rover maps can be subject to various operations before being displayed to users:

Filter: Objects in a Rover map have a set of attributes that identify certain properties of the objects. Depending on the user's context (which indicates the user's interests), filters are

www.studymafia.org

generated for the attribute values of interest to the user. These filters are applied to maps to select the appropriate subset of objects to display to the user. For example, one user may be interested in only the restaurants in a specific area, while another user needs to view only the museum and exhibition locations. The filters can be dynamically changed to appropriately change the objects being displayed on the map.

Zoom: The zoom level of a displayed map identifies its granularity. The zoom level at a client device is chosen based on the user's context. For example, a user inside a museum gets a detailed museum map, but when the user steps outside the museum, he gets an area map of all the museums and other points of interest in the geographic vicinity. The zoom level can be implemented as an attribute of objects, and appropriate filters can then be applied to display a map at the desired zoom level.

Translate: This functionality enables the map service to automatically update the view of the displayed map on the client device as the user moves through the system. When the location of the user moves out of the central region of the currently displayed map, the system prepares a new map display, which is appropriately translated from the previously displayed map.

ROVER ARCHITECTURE

A Rover system, depicted in Figure 1, consists of the following entities:

End-users of the system. Rover maintains a **user profile** for each end-user, that defines specific interests of the user and is used to customize the content served.

Rover-clients are the client devices through which users interact with Rover. They are typically small wireless handheld units with great diversity of capabilities in regard to processing, memory and storage, graphics and display, and network interface. Rover maintains a *device profile* for each device, identifying its capabilities and thus, the functionality available at that device.

Wireless access infrastructure provides wireless connectivity to the Rover clients. Possible wireless access technologies include IEEE 802.11 based wireless LANs, Bluetooth and Cellular services. For certain Quos guarantees, additional mechanisms need to be implemented at the access points of these technologies for controlled access to the wireless interface.

Servers implement and manage the various services provided to the end-users. The servers consist of the following:

www.studymafia.org

– **Rover controller**: is the "brain" of the Rover system. It provides and manages the different services requested by the Rover clients. It schedules and filters the content sent to the clients based on use and device profiles and their current locations.

– **Location server**: is a dedicated unit responsible for managing the client device location services within the Rover system. Alternatively, an externally available location service can also be used.

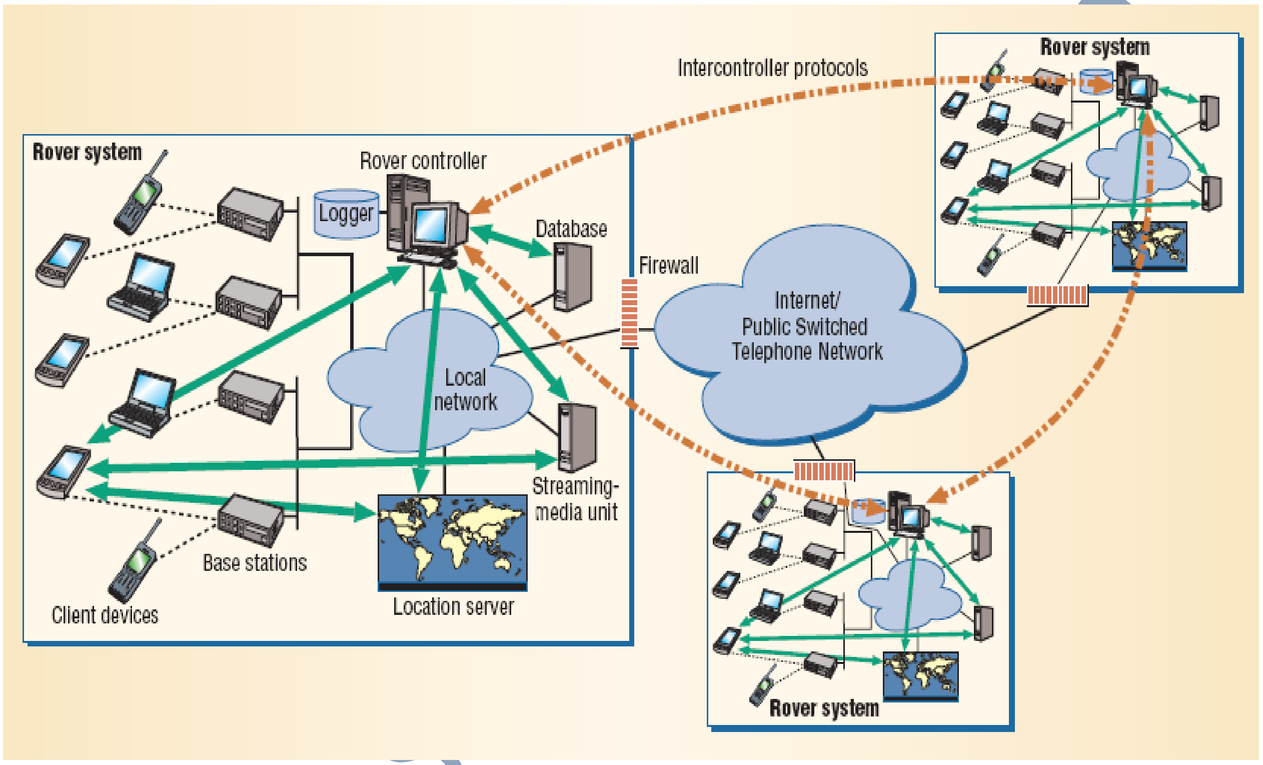


Fig 3.1 Physical architecture of the Rover System

– **Media streaming unit**: provides the streaming of audio and video content to the clients. In fact, it is possible to use many of the off-the-shelf streaming-media units that are available today and integrate them in the Rover system.

– **Rover database**: stores all content delivered to the Rover clients. It also serves as the stable store for the state of the users and clients that is maintained by the Rover controller.

– **Logger**: interacts with all the Rover server devices and receives log messages from their instrumentation modules.

<u>System Scalability</u>

www.s

There are two potential bottlenecks that can hinder the scalability of such a system to large user populations. One is the server system because it needs to handle a very large number of client requests with tight real-time constraints. Another potential bottleneck is the bandwidth and latency of the wireless access points.

For a server to handle such a large volume of real-time requests, in addition to adequate compute power and appropriate data structures, it must have **fine-grained real-time application-specific scheduling** of tasks to efficiently manage the available resources, both processing and bandwidth. This leads us to divide server devices into two classes:-

- **Primary servers**, which directly communicate with the clients, and
- **Secondary servers**, which do not directly communicate with clients but interact with primary servers to provide backend capabilities to the system.

The Rover controller, location server and media streaming unit are examples of primary servers, while the Rover database and the logger are examples of secondary servers.
In order to meet the performance objectives, only the primary servers need to implement the fine-grained real-time task scheduling mechanism. We have defined a concurrent software architecture called the *Action model* that provides such a scheduling mechanism, and implemented the Rover controller accordingly. The Action model, explained below, avoids the overheads of thread context switches and allows a more efficient scheduling of execution tasks. The Rover system exports a set of well defined interfaces through which it interacts with the heterogeneous world of users and devices with their widely varying requirements and capabilities. Thus new and different client applications can be developed by third-party developers to interact with the Rover system.

A Rover system represents a single domain of administrative control that is managed and moderated by its Rover controller. A large domain can be partitioned into multiple administrative domains each with its own Rover system, much like the existing Domain Name System [9]. For this multi-Rover system, we define protocols that allow interaction between the domains. This enables users registered in one domain to roam into other domains and still receive services from the system

www.studymafia.org

ACTION MODEL

In order to achieve fine-grained real-time application-specific scheduling, the Rover controller is built according to concurrent software architecture we call the **action model**. In this model, scheduling is done in “atomic” units called actions. An **action** is a “small” piece of code that does not have any intervening I/O operations. Once an action begins execution, it cannot be pre-empted by another action. Consequently, given a specific server platform, it is easy to accurately bound the execution time of an action. The actions are executed in a controlled manner by an Action Controller.

We use the term **server operation** to refer to a transaction, either client- or administrator-initiated, that interacts with the Rover controller; examples in the museum scenario would be *register Device*, *get Route* and *locate User*. A server operation consists of a sequence (or more precisely, a partial order) of actions interleaved by asynchronous I/O events. Each server operation has exactly one “response handling” action for handling all I/O event responses for the operation; i.e., the action is eligible to execute whenever an I/O response is received.

A server operation at any given time has zero or more actions eligible to be executed. A server operation is in one of the following three states:

- Ready-to-run: At least one action of the server operation is eligible to be executed but no action of the server operation is executing.

-Running: One action of the server operation is executing (in a multi-processor setup, several actions of the operation can be executing simultaneously).

-Blocked: The server operation is waiting for some asynchronous I/O response and no actions are eligible to be executed.

The Action Controller uses administrator-defined policies to decide the order of execution of the set of eligible actions. The scheduling policy can be a simple static one, such as priorities assigned to server operations, but it can equally well be time based, such as earliest-deadline-first

www.studymafia.org

or involving real-time cost functions. In any case, the controller picks an eligible action and executes it to completion, and then repeats, waiting only if there are no eligible actions (presumably all server operations are waiting for I/O completions).

The management and execution of actions are done through a simple **Action API** defined as follows:

-*init (action id, function ptr)*: This routine is called to initialize a new action (identified by *action id*) for a server operation. *Function ptr* identifies the function (or piece of code) to be executed when the action runs.

-*run (action id, function parameters, deadline, deadline failed handler ptr):* This routine is called to mark the action as eligible to run. *Function parameters* are the parameters used in executing this instance of the action. *Deadline* is optional and indicates the time (relative to the current time) by which the action should be executed. This is a soft deadline, that is, its violation leads to some penalty but not system failure. If the action controller is unable to execute the action within the deadline, it will execute the function indicated by *deadline failed handler ptr*. This parameter can be *NULL*, indicating that no compensatory steps are needed.

-*cancel (action id, cancel handler ptr)*: This routine is called to cancel a ready-to-run action provided it is not executing. *Cancel handler ptr* indicates a cleanup function. It can be *NULL*.

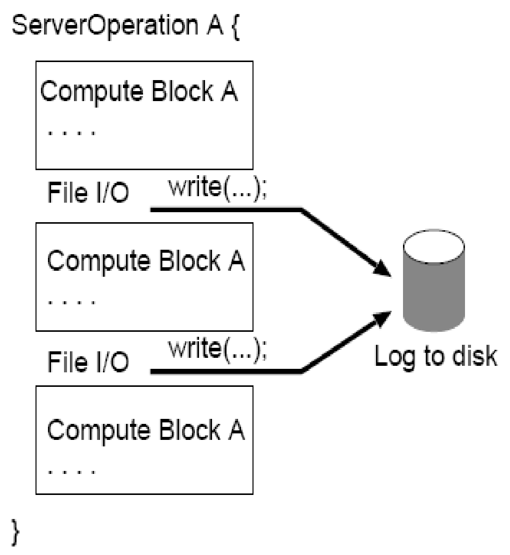
Actions vs. Threads:

Our need to scale to very large client populations made us adopt the action model rather than the more traditional thread model. We now provide some experimental justification. There are several ways to use a thread model to implement the Rover controller. One is to implement each server operation as a separate thread. Another is to have a separate thread for each user. Both of these imply a large number of simultaneously active threads as we scale to large user populations, resulting in large overheads for thread switching. A more sensible approach is to create a small set of "operator" threads that execute all operations, for example, one thread for all *registerDevice* operations, one for all *locateUser* operations, and so on. Here

www.studymafia.org

the thread switching overhead is modest but there are drawbacks. One is that, depending on the threads package, it restricts our ability to optimize thread scheduling, especially as we transit to time-based (rather than priority based) scheduling. More importantly, because each operator thread executes its set of operations in sequence, this approach severely limits our ability to optimally schedule the eligible actions within an operation

and across operations. Of course, each thread could keep track of all its eligible actions and do scheduling at the action level, but this is essentially recreating the action model within each thread.



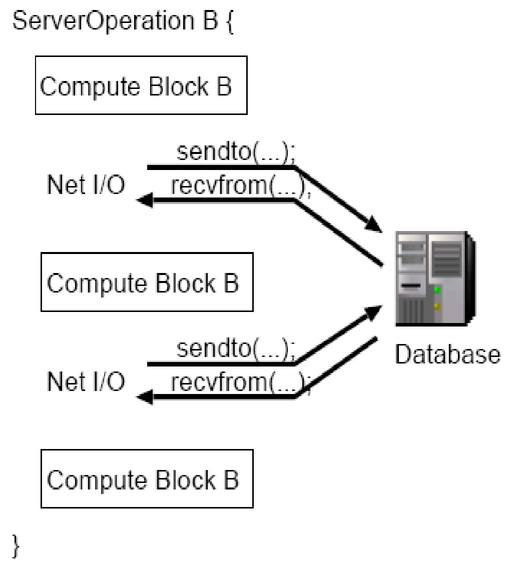


Figure 3.2: Scenario A has 10,000 processor-bound server
server operations where computation is inter-
interleaved
leaved with file write operations

Figure 3.3: Scenario B has 100 I/O bound server
operations where computation is
with network I/O interactions

www.studymafia.org

We compare the performance of action-based and thread-based systems through implementation. In this paper, we consider two kinds of server operations, one processor-bound and the other, I/O-bound, both of which appear in the context of the Rover controller. Using these server operations, we construct two corresponding scenarios:

1. *Scenario A:* This is a computation-intensive scenario and has 10,000 processor-bound server operations, where each of the server operations has three compute blocks, interleaved with two file write operations (see Figure 3.2). In each of these server operations, the second and the third I/O compute block do not need to await for the prior file I/O write operation to complete.

2. *Scenario B:* This is an I/O-intensive scenario and has 100 I/O-bound server operations, where each of the server operations has three compute blocks, interleaved with two network I/O operations (see Figure 3.3). In each of these server operations, the second and third compute blocks can be initiated only after the completion of the prior network I/O operation. The network I/O interaction was implemented using UDP. Since our focus is on the comparison of the action-based versus the thread-based systems, we avoided issues of packet loss and re-transmissions by only considering those experiments where no UDP packets were lost in the network.

We consider two execution platforms, referred to as M1 and M2 in the paper. M1 comprises of a Intel Pentium III (600 MHz) processor and 96 MB of RAM which runs Linux. M2 comprises of a Sun Ultra 5, with a Sparc (333 MHz) processor and 128 MB RAM and runs Solaris. For the thread-based implementation, we used the Linux Threads library for the M1 platform and the Pthreads library for the M2 platform, both of which are implementations of the Posix 1003.1c threads package. This total execution time for the three compute blocks in each server operation A was 0.1518 ms for M1 and 0.9069 ms for M2. The ping network latency for the network I/O in server operation B varied between 30-35 ms.

We compared performances of an action-based implementation and a thread-based implementation of the two scenarios for the different platforms. In the action-based implementation each compute block is implemented as a separate action. In the thread-based implementation, we experimented with a different number of threads, where each thread executed an equal number of server operations for perfect load balancing between the different threads.

www.studymafia.org

Table 1: Comparisons of overheads for action-based and thread-based systems (in ms)

Scenario	Machine Specifications	Action-based	Thread-based (threads used)				
			1	5	10	50	100
A (Fig. 2) M1	Pentium/Linux	**24.27**	299.3	299.9	300.4	304.5	310.3
A (Fig. 2) M2	Sparc/Solaris	**62.82**	1000.9	1012.5	1041.6	1012.8	1031.25
B (Fig. 3) M1	Controller, M2 Database	**11.61**	3711.9	1302.2	1011.4	893.10	728.30

In Table 1, we present the overheads obtained in each case, where the overhead is the total execution time minus the fixed, identical and unavoidable computation/communication costs for the two scenarios. We report the mean execution overheads of a large number of runs, which were required to obtain low variance. For the computation-intensive server operations there are very little performance gains in trying to overlap computation with communication (file I/O), and is not substantial enough to justify the overheads of a multi-threaded implementation. Therefore, a thread-based system with a single thread achieves the best performance among the thread-based implementations.

For the I/O intensive server operations, using a multi-threaded implementation is useful, since computation and communication can be overlapped. Consequently, the best performance for the thread-based system is achieved, when the maximum number of threads is used (one thread for each server operation).

As can be observed in both scenarios, the action-based implementation still achieves significantly (about an order of magnitude) less overhead as compared to the best thread-based implementation.

www.studymafia.org

ROVER CLIENTS

The client devices in Rover are handheld units of varying form factors, ranging from powerful laptops to simple cellular phones. They are categorized by the Rover controller based on attributes identified in the device profiles, such as display properties—screen size and color capabilities, text and graphics capabilities, processing capabilities — ability to handle vector representations and image compression, audio and video delivery capabilities and user interfaces. The Rover controller uses these attributes to provide responses to clients in the most compatible formats.

For the wireless interface of client devices, we have currently considered two link layer technologies — IEEE 802.11 Wireless LAN and Bluetooth. Bluetooth is power efficient and is therefore better at conserving client battery power. According to current standards, it can provide bandwidths of up to 2 Mbps. In contrast, IEEE 802.11 wireless is less power-efficient but is widely deployed and can currently provide bandwidths of up to 11 Mbps. In areas where these high bandwidth alternatives are not available, Rover client devices will use the lower bandwidth air interfaces provided by cellular wireless technologies that use CDMA [11] or TDMA based techniques. In particular, cellular phones can connect as clients to Rover, which implies that the Rover system interfaces with cellular service providers.

Different air-interfaces may be present in a single Rover system or in different domains of a multi-Rover system. In either case, *software radios* [8] is an obvious choice to integrate different air-interface technologies. While the location management system is not tied to a particular air interface, certain properties of specific air interfaces can be leveraged to better provide location management (discussed in the Appendix).

www.studymafia.org

ROVER CONTROLLER

The interaction of the Rover controller with all other components of the system is presented in Figure 4. The Rover controller interacts with the external world through the following interfaces:

Location Interface: This interface is used by the Rover controller to query the location service about the positions of client devices. The location of a device is defined as a tuple representing the estimate of its position (either absolute or relative to some well-known locations), the accuracy of the estimate, and the time of location measurement. Depending on the technology being used to gain position estimates, The accuracy of the estimate depends on the particulars of the location technology, for example, GPS [6], IEEE 802.11 signal strength, signal propagation delays, etc. Rover takes into account this accuracy information when making location-based decisions.

Admin Interface: This interface is used by system administrators to oversee the Rover system, including monitoring the Rover controller, querying client devices, updating security policies, issuing system specific commands, and so on.

-*Content Interface:* This interface is used by the content provider to update the content that is served by the Rover controller to the client devices. Having a separate content interface decouples the data from the control path.

Back-end Interface: This interface is used for interaction between the Rover controller and certain external services as may be required. One such service is e-commerce, by which credit card authorization for various purchases can be made. These services would typically be provided by third-party vendors.

Server Assistants Interface: This interface is used for interaction of the Rover controller with the secondary servers. e.g. the database and the streaming media unit.

www.studymafia.org

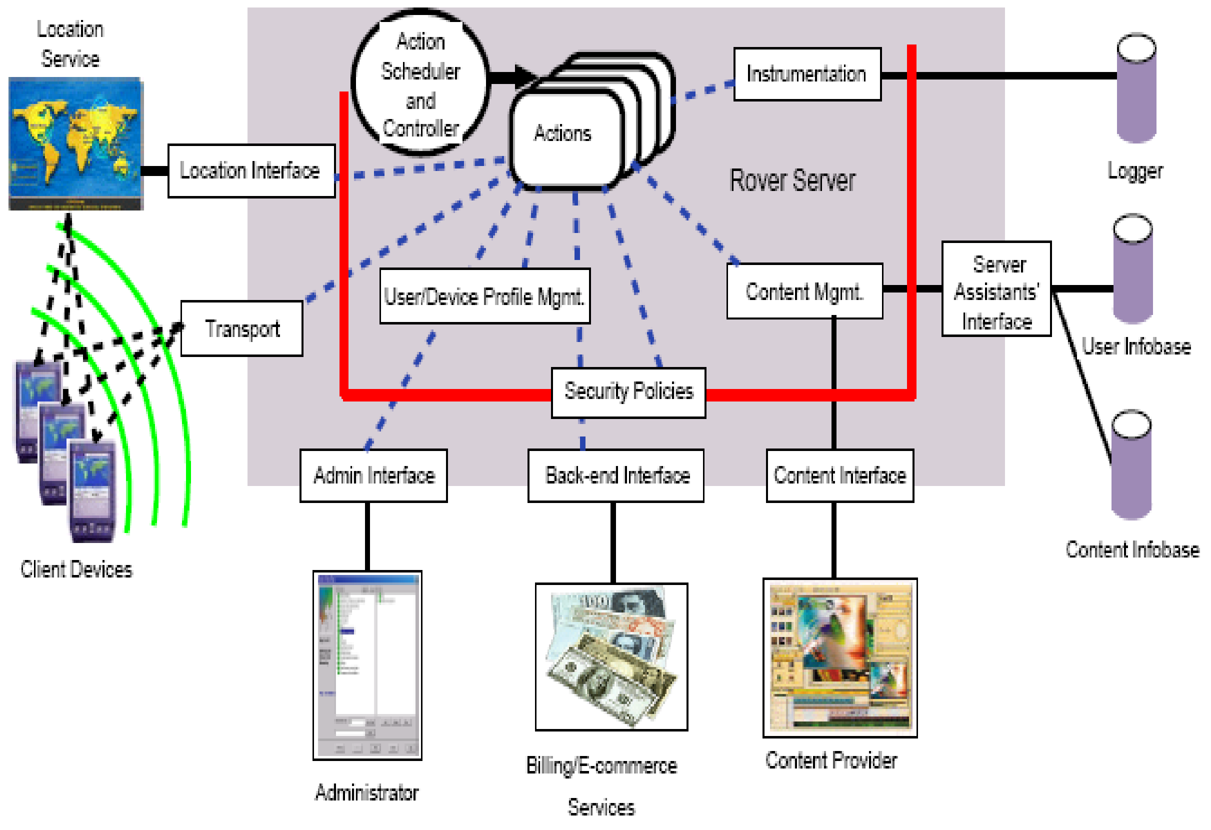


Fig 3.4 Logical architecture of a Rover system

www.stud

Transport Interface: This is the communication interface between the Rover controller and the clients, which identify data formats and interaction protocols between them.

ROVER DATABASE

The database in Rover consists of two components, which together decouple client-level information from the content that is served.

One component of the database is the **user info base**, which maintains user and device information of all active users and devices in the system. It contains all client-specific contexts of the users and devices, namely profiles and preferences, client location, and triggers set by the clients. This information changes at a fairly regular rate due to client activities, e.g. the client location alters with movements. The Rover controller has the most updated copy of this information and periodically commits this information to the database. For many of these data items (e.g. client location), the Rover controller lazily updates the database. These are termed as *volatile data* since any change to these data items are not guaranteed to be accurately reflected by the system across system crashes. For some others, (e.g. new client registration) the Rover controller commits this information to the database before completing the operation. These are termed as *non-volatile data*. The Rover controller, identifies some parts of the data to be volatile, so as to avoid very frequent database transactions. The Rover controller does not guarantee perfect accuracy of the volatile data, and thus trades off accuracy with efficiency for these data components.

The other component in the database is the **content info base**. This stores the content that is served by the Rover controller and changes less frequently. The content provider of the Rover system is responsible for keeping this info base updated. In the museum example, this component stores all text and graphical information about the various artifacts on display.

The Rover database implements an extended-SQL interface that is accessed by the Rover controller. Apart from the usual SQL functionality, it also provides an API for retrieval of spatial information of different objects and clients in the system.

The transactions of the Rover controller with the database are executed on behalf of the different server operations. The transactions, by definition, are executed atomically by the database. Additionally, each transaction is identified by two different flags that identify certain properties for execution, as follows:

Lock-Acquiring: If this flag is set, the transaction is required to acquire relevant locks, on behalf of the server operation, to read or write data to the database. It also requires that these locks will be released by the server operation prior to its termination at the Rover controller.

www.studymafia.org

Blocking: If a transaction issued by a server operation is unable to access or modify some data due to locks being held by other server operations, it can either block till it successfully reads the data, or it returns immediately to the server operation without successfully execution. If the Blocking flag is set for a transaction, then the first option is chosen for the transaction.

To avoid deadlocks, server operations acquire the relevant locks on data items stored in the database using a *Two Phase Locking* protocol with a lexicographic ordering of lock acquiring for data items. It is important to note that server operations may need to acquire locks at the database, if and only if they need to access the stored data through multiple transactions and all these transactions need to have the same data view. This is not required for the vast majority of server operations that either make a single database transaction, or do not need its multiple transactions to have identical views. None of the server operations in the current implementation of Rover, required to acquire locks at the database. The transactions themselves might acquire and release locks at the database during their execution, which are not visible to the server operations at the Rover controller.

LOCATION SERVER

The location server is responsible for storing and managing user locations in the Rover system. The system is designed to work in both indoors and outdoors environments. We have experimented with RF-based systems that infer the location of a device based on the signal strength of received RF signals of IEEE 802.11 wireless LAN frames.

In our RF-based techniques, the user location of a client is obtained without the use of any additional hardware. It thus provides more ubiquitous coverage in campus-like environments that already have a rich wireless LAN coverage for data transport. This can be contrasted to alternative Infra-red tag-based systems [18, 11, 3] or ultra-sonic emitter and receiver based systems [16] in which additional devices need to be attached to the infrastructure as well as the clients. We have developed different RF-based technique in the context of the Rover system. Techniques are categorized into:

www.studymafia.org

– *Radio-map Techniques:* Work in 2 phases: an offline phase and a location determinationphase. During the offline phase, the signal strengths received from the access points, at selected locations in the area of interest, are gathered as vectors and tabulated over the area. During the location determination phase, the vector of samples received from each access point is compared to the radio-map and the ”best” match is returned as the estimated user location. We used two methods to calculate the best match:

- K-Nearest Neighbors (KNN): A distance function is defined to measure the distance between any two data vectors. The nearest K vectors to the vector of samples, received from each access point at the location determination phase, are calculated. Then from the K vectors a vote is conducted to estimate the best user location.
- Probabilistic Clustering-based: Baye’s theorem is used to estimate the probability of each location within the radio-map. Then the most probable location, using the vector of samples, is reported as the estimated user location. Refer to [21] for more details

– *Model-based Techniques:* The relation between the signal strength received from an access point and the distance to this access point is captured by some function (model). By using three or more access points, the user location is estimated. Two methods are used:

- Minimum Triangulation: Given each access point *i* located at coordinates *xi; yi; zi*, the distance between the receiver and the access point (*di*) is modeled as:

    - $$d_i^2 = (x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = \frac{k}{v_i}$$

  where *x; y; z* are receiver’s coordinates, *vi* is the strength of the received signal, and *k* is a constant. Due to problems in signal propagation like reflection and multi pathes,we model the problem as finding a solution to the minimization problem:

www.studymafia.org

- $$\min f(x,y,z,k) = \min \sum_{i=1}^{n} (v_i d_i^2 - k)^2$$

  By solving the derivatives equations of *f* numerically, we can get the estimation coordination of the receiver.

- Curve Fitting: The received signal power is modeled as:

$$PL(d)[dB] = A + B\log(d)$$

- $$A = PL(d_0)[dB] - 10n\log(d_0),\ and\ B = 10n$$

  where PL is the received power at certain position in decibels, d is the three dimensional path length between the transmitter and the receiver, *d*0 is a reference distance, and *n* represents the path loss exponent. We estimate the *A* and *B* parameters for each access points using curve fitting techniques. Given the vector of samples, we can estimate how far the receiver is from each access point to estimate his location.

For indoor environments, we found that radio-map based techniques achieve better accuracy than model-based techniques. This is because the relation between the signal strength hand distance in indoor environments is complicated by to the multi-path effect and other phenomena which are difficult to capture by simple models. On the other hand, model based techniques have the advantage of not depending on the calibration process required to build the radio-map. This advantage favors model-based techniques in outdoor environments, where the relation between signal strength and distance can be captured by simple functions and the coverage area is large making building the radio-map a time consuming process.

MULTI-ROVER SYSTEM

A single Rover system comprises of a single Rover controller, other server devices (e.g., Rover database and Rover streaming media unit), and a set of Rover clients. A single system is sufficient for management of Rover clients in a zone of single administrative control. For example, consider a Rover system in a single museum. All artifacts and objects on display in the

www.studymafia.org

museum are managed by a single administrative entity. There is a single content provider for this system and a single Rover system is appropriate to serve all visitors to this museum.

However, each separate museum has its independent administrative authority. Therefore, we can have a separate Rover system for each of the different museums that are administered separately by each museum authority. This allows a decentralized administration of the independent Rover systems, locally by each museum authority. However, it is important to provide a seamless experience to visitors as they roam from museum to museum. A multi-Rover system is a collection of independent Rover systems that peer with each other to provide this seamless connectivity to the user population.

The design of a multi-Rover system is similar in spirit to the Mobile IP [10] solution to provide network layer mobility to devices. Each client device has a *home* Rover system to which it is registered. As the device physically moves into the zone of a different, or *foreign* Rover system, it needs to authenticate itself with the Rover controller of the foreign system. Based on administrative policies, the two Rover systems have service level agreements that define the services that they will provide to clients of each other.

When the Rover controller of a system detects a foreign client device, it first checks whether it has an appropriate service-level agreement with the Rover controller of the device's home system. If one exists, the Rover controller of the foreign system requests transfer of relevant state about the client device from the Rover controller of the home system and subsequently provides necessary services to it. Rover controllers of different Rover system use the Inter-Controller protocols to interact.

www.studymafia.org

ROVER IMPLEMENTATION AND ANALYSIS

COMMUNICATION INTERFACES

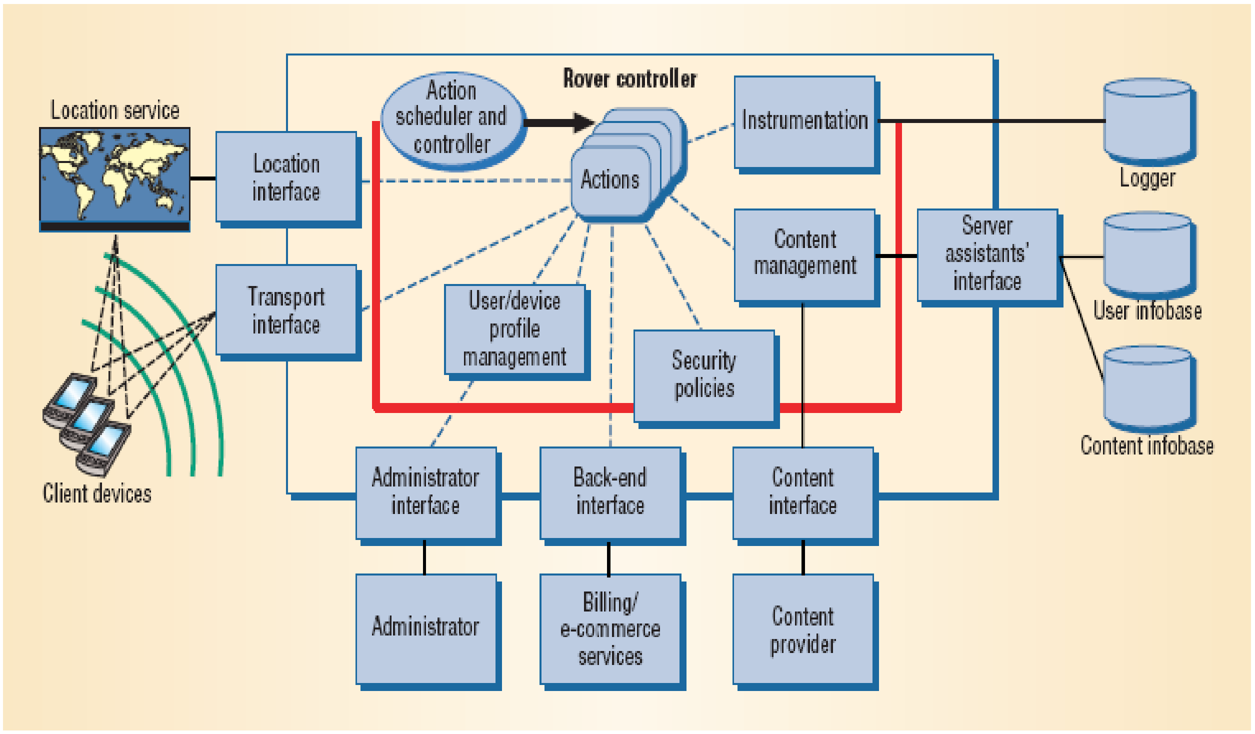


Fig 4.1 Rover's controller interacts with other parts of the system and the external world

www.st

For the wireless interface to client devices, we considered two link-layer technologies: IEEE 802.11 and Bluetooth. Bluetooth is power efficient and therefore better at conserving client battery power. According to current standards, Bluetooth can provide bandwidths up to 2 Mbps. In contrast, IEEE 802.11 is less power efficient but widely deployed and currently provides bandwidths up to 11 Mbps. In areas where these high-bandwidth alternatives are not available, Rover client devices will use the lower bandwidth interfaces that cellular wireless technologies provide. Figure 4.1 shows how Rover's controller interacts with other parts of the system and with the external world. The controller uses the *location interface* to query the location service about the positions of client devices and the *transport interface* to identify data formats and interaction protocols for communicating with the clients. It uses the *server assistants' interface* to interact with secondary servers like the database and the streaming media unit and the back-end interface to interact with external services, such as credit card authorization for e-commerce purchases. Third-party providers typically offer these external services. System administrators can use the *admin interface* to oversee the Rover system, including monitoring the Rover controller, querying client devices, updating security policies, issuing system-specific commands, and so on. The *content interface* lets content providers update the information and services that the Rover controller serves to client devices. Having a separate content interface decouples the data from the control path.

www.studymafia.org

INITIAL IMPLEMENTATION

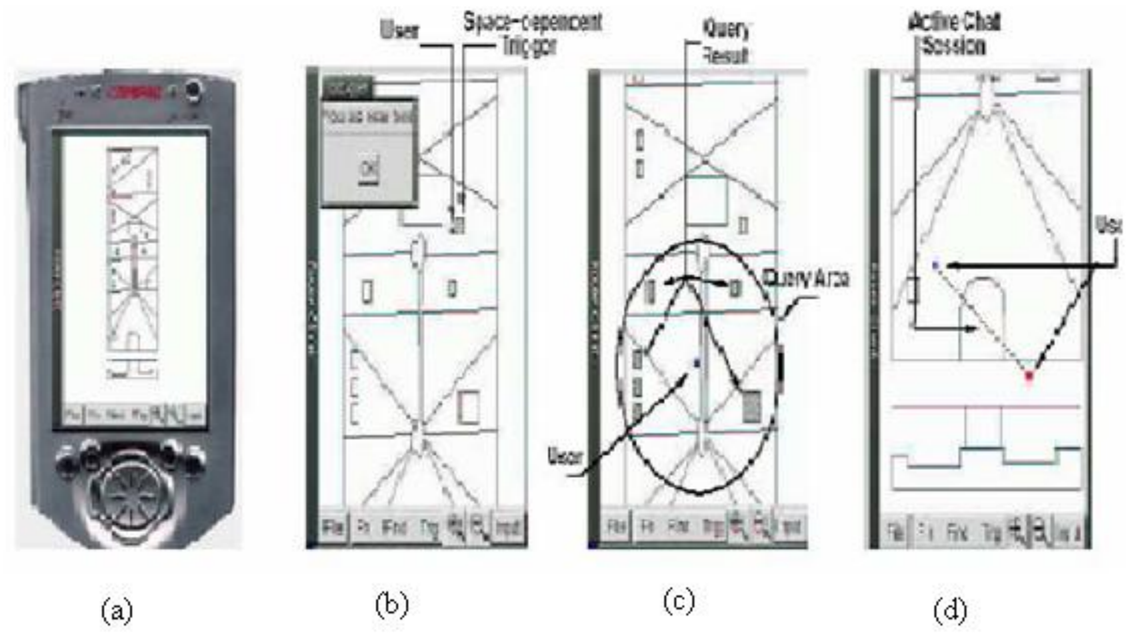


Fig 4.2 Rover client screen shots taken from a demonstration at the McKeldin mall of the University of Maryland campus. (a) Rover client running the client software showing the mall map. (b) A notification to the client about a nearby food stall. The user associated with the client had previously set a trigger notification request when he is close to a food stall. (c) The user had issued a query operation about the sites of interest in his vicinity. On receiving the response from the Rover system, the client has highlighted the relevant sites. (d) An active chat session between this user and another user is marked as a dotted line connecting both users.

We have successfully built Rover prototype systems and tested them in the campus of University of Maryland College Park. The implementation has been demonstrated for both indoor and outdoor environments. A preliminary test implementation was developed on Windows based systems (Windows 2000 for the controller and Windows CE for the client devices). The current implementation of the Rover system has been developed under the Linux operating system. The Rover controller is implemented on a Intel Pentium machine running Red Hat Linux 7.1 and the clients are implemented on Compaq iPAQ Pocket PC (model H3650) running the Familiar distribution (release versions 0.4 and 0.5) of Linux for PDAs1. Wireless access is provided using IEEE 802.11 wireless LANs. Each Compaq iPAQ is equipped with a wireless card which is attached to the device through an expansion sleeve.

We have experimented with a set of 8 client devices and have tested various functionalities of the system.

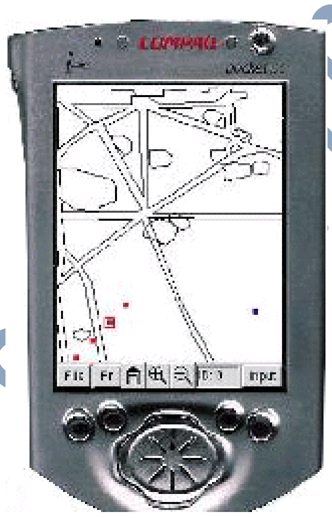


Figure 4.3: View of the display of a Rover-client

For our outdoor experiments, we interfaced a GPS-device (Garmin e-Trex) to the Compaq iPAQs and obtained device location accuracy of between 3-4 meters. The display of the iPAQ Rover-client displays the locations of the different users (represented by the dots) on the area map as shown in Figure 4.3. The indoor Rover system is implemented for the 4th floor of the A.V.Williams Building (where the Computer Science Department is located), whose map is shown in Figure 6. In this implementation, the location service is being provided using signal

www.studymafia.org

strength measurements from different base stations. There are about base stations that are distributed all over the building and typically the client device can receive beacons from five or six of the base stations. We are able to get an accuracy of better than a meter in this environment, using very simple signal-strength based estimation techniques.

In both these cases, we implemented the basic functionality of the Rover system. They include:

- User activation/de-activation and device registration/de-registration procedures.
- Periodic broadcast of events of interest from the Rover controller to the users in specific locations.
- Interaction between users. This can be either simple text messaging or voice chat. Users can optionally make their location visible to other users. In the museum example, a tour group coordinator can use this feature to locate all the other members of the group.
- Users can request alerts from the Rover controller when certain conditions are met. The conditions may be time, location or context dependent. This can be used to provide notification to ticket holders of an approaching show time. Clearly, for the users who are further away from the show venue, this notification needs to be provided early enough, so that they have enough time to reach the venue.
- An administrator's console allows a global view of all users and their locations in the system. The administrator can directly interact with all or a specific subset of the users based on the location or other attributes of the users.

Currently, we are implementing more functionality in the Rover controller.

www.studymafia.org

System Functionality

The Rover system provides different capabilities to the users, which can be categorized as follows:

- System Admin Operations are available only to the authorized system administrator. These set of operations are—register new user/device, update user/device attributes and de-register user/device. The administrator is also able to query the Rover server system about the state and information specific to any and all Rover clients in the system.

- User Access Operations are the basic set operations that every user avails to access theRover system. They include the user login and user logout operations.
- Trigger Operations allow users to set context-specific alerts. The triggers are activated based on user interests and depend on current time and/or location of the user. An user can enable triggers by specifying the relevant time or space-dependent condition. When the trigger condition is satisfied the Rover server system sends appropriate notification to the particular user (Figure 2(b)).
- Query Operations allow users to acquire information about different aspects of the system and the environment. For example, an user can request information about all or a subset of all active users in the system. Figure 2(c) shows a client screen shot in response to a client query on sites of interest in its vicinity.
- Location Update Operation inform the server system about the client's location using.
- Audio Chat Operations enable direct audio communication between clients. Audio chat between clients is initiated with the coordination of the Media Manager. Once an audio chat is initiated, the clients interact directly with each other without intervention of the rover server system. Figure 2(d), shows the display at a client that is involved in anaudio chat with another client. The dashed line indicates an active chat session between the clients 4.
-

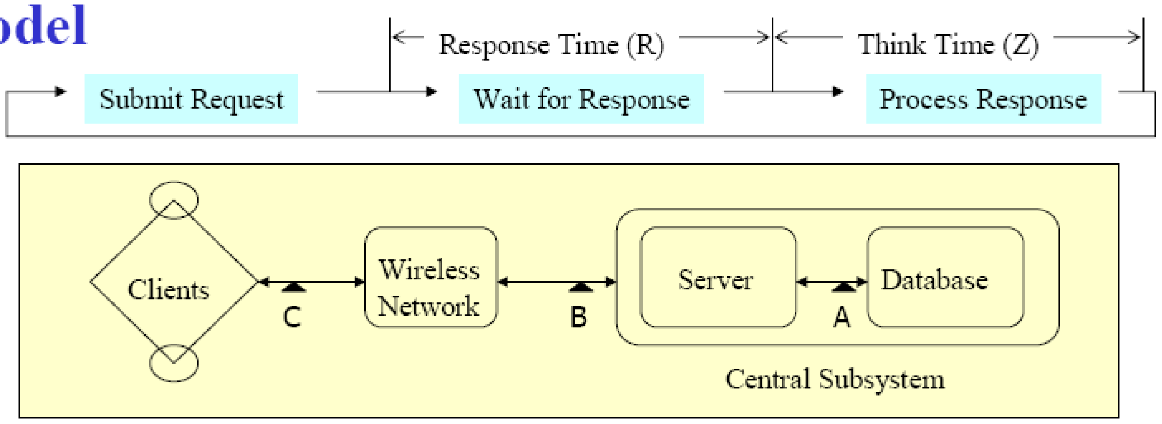
SYSTEM PERFORMANCE

To assess the performance and scalability of the Rover System we take two approaches:
a) Active Monitoring where we instrumented the controller to collect different performance statistics (e.g. queue lengths for each component, the response time for each operation, etc.).

b) Passive Monitoring which is described in on next page.

www.studymafia.org

Model



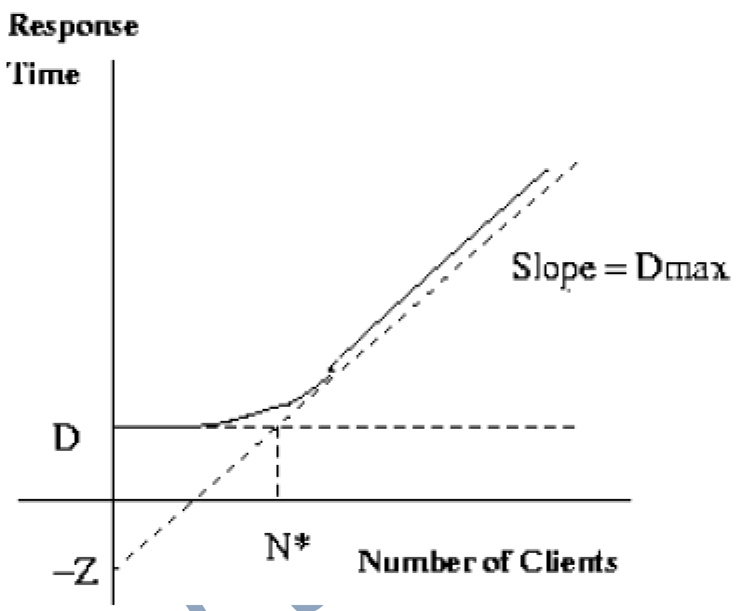


Fig. 4.4. Passive monitoring analysis: (a) Performance model for passive monitoring. (b) Typical asymptotic bounds.

www.studymafia.org

Analysis using Passive Monitoring

In this approach, no instrumentation code is introduced in the server system. Instead, we use a client load generator to stress test the server and observe two different metrics — the response time obtained by individual clients and the number of clients that can simultaneously served by the system without significantly impacting the performance of the clients.

We model the Rover system as a single-server multi-client system as shown in Figure 4.4(a). The Rover System is modeled as a central subsystem consisting of two devices, the Rover controller and the Rover database, and N terminal subsystems. Each of the N terminals is a client of the Rover System and perform the cycle of issuing a request, waiting for the response and processing the response (think time). Wireless network models the communication channel between the server and the clients. Since we are interested in assessing the performance of the Rover system we do not explore the affects of communication channel in this paper.

Using a technique called *operational law* [10] to analyze such systems, it can be shown that the response time observed by clients increase marginally with increasing the number of clients up to a critical client population. Let *D* denotes the time required by the system to process a single client operation, and *D*max denotes the time required at the bottleneck server of a multi-server system. For the single server model, *D*max = *D*. If *N** indicates the critical number of clients that the system can support without impacting the response time for the clients and *Z* the *think time* used by the clients between operations, then operational analysis suggests that:

$N^* = (D + Z)/D\text{max}$

The graphical representation of *N** is shown in Figure 4.4(b).

<u>Experiment Configuration</u>

The central subsystem runs on Pentium IV 1.5 GHz desktop machine with 256 MB of RAM running the Linux OS with kernel version 2.4.7. A second machine is used to behave as a set of clients (client loader). The client loader runs on a Pentium III 800 MHz laptop with 128 MB of RAM and running a Linux OS with kernel version 2.4.2. The client machine uses 802.11b wireless network to connect to the network.

The response time for each operation were collected as observed at the database, the controller and the client (points A, B, C respectively in Figure 3(b)). Instead of collecting response time for each of the system operations, we experimented with three different operations representing three different categories:

1. *GetAllLoginUsers:* Gets the position of all users who are logged into the system. This operation is controller intensive and does not involve the database.

2. *VectorMap:* Gets the vector map of an area. This operation is computationally intensive at the database side.

www.studymafia.org

www.studymafia.org

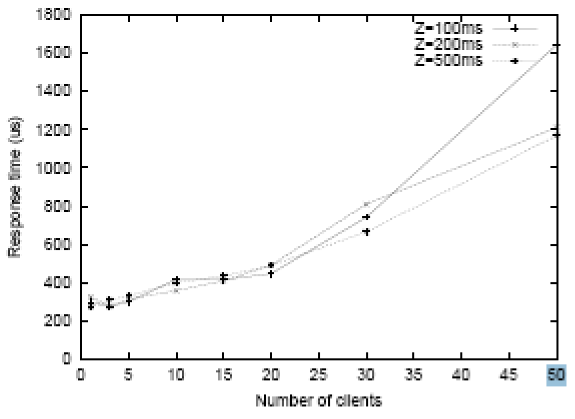
3. *Locate:* Locates the object containing the given point. This operation involves the database though it is not very computationally intensive at either the database or the controller sides.

<u>Results and Discussion</u>

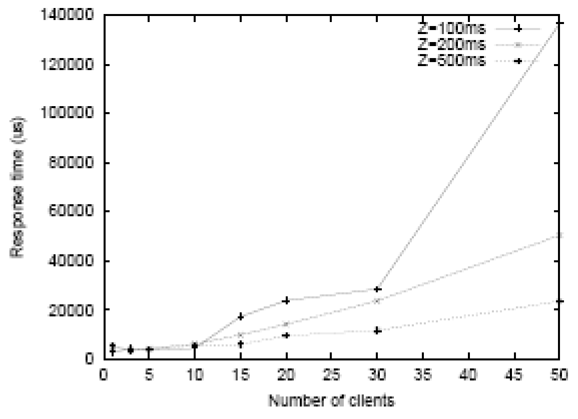
In this section we show the results obtained for each of the above operations.

<u>GetAllLoginUsers</u>

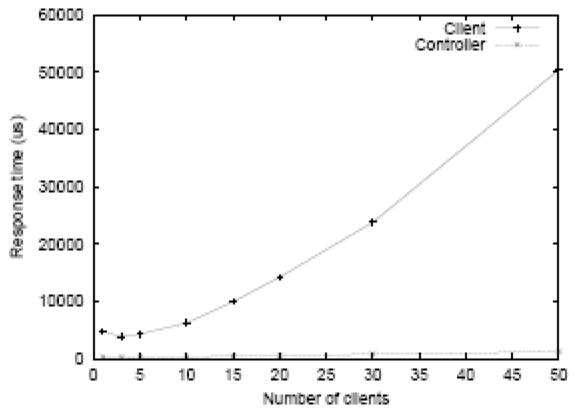
Figure 4.5(a) shows the response time at the controller plotted against the number of clients in the system for different think times (*Z* = 100*ms;* 200*ms;* 300*ms*). The total



(a)



(b)



(c)

Fig 4.5 *GetAllLoginUsers* operation: (a) Controller response time, (b) Client response time, and (c) Response timewhen Z=200ms.

service time, *D* of the controller is observed to be around 300 microseconds5. For only one device in the system *Dmax* = *D*. Using Equation 1 where *D* = 300 microseconds and a think time *Z* = 200 milliseconds, we get *N*¤ to be approximately 667 requests. Hence the server can support 667 requests without any significant delays. In an actual deployment, the think time would be of the order of 10's of seconds and that would give an even higher value of *N**(of the order of thousands of clients). Figure 4.5(b) shows the response time for the client side.Figure 4.5(c) shows the response time behavior of both the controller and the client when the think time is Z = 200 milliseconds. As the graph shows the controller graph stays almost horizontal as the number of clients are increased which shows the controller can handle a large number of clients. On the other hand, the client graph grows with the number of clients. This can either be due to the effect of the wireless hop involved or the processing involved at the OS level. The performance of the 802.11b wireless network has not been taken into account and is left for future work.

VectorMap

Figure 4.6 shows the response time at the different Rover components. For the database the total service demand time is observed to be around 0.5 seconds. Using equation 1, we can predict the knee-point to be at *N*¤ = 3 with a think time *Z* = 1 second. We should note that the *VectorMap* operation is an infrequent operation and has been used only to assess the performance of the system in the extreme case. In an actual deployment, the duration between subsequent *VectorMap* operation requests (Z) would be in the order of minutes.

Figure 4.6 shows the response time observed at the database, the controller, and the client when the think time is *Z* = 1 second. The difference in the database response and the controller response could be explained by the fact that at the controller all the data is touched and a copy is created for debugging purpose. Once this debugging code is taken out the controller graph should follow the database graph very closely.

www.studymafia.org

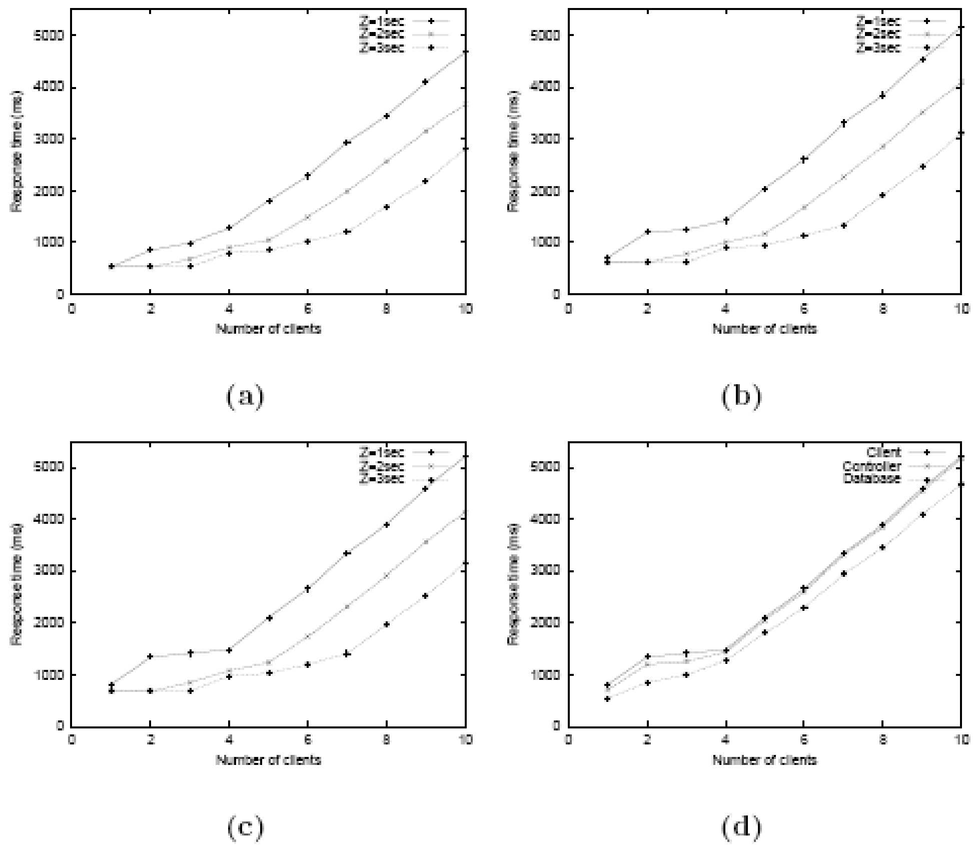


Fig. 4.6. *VectorMap* operation: (a) Database response time, (b) Controller response time, (c) Client response time, and(d) Response time when Z=1s.

www.stu

Locate

Similar to the analysis of the previous operations, we show the response time at the database, the server and the client for a think time of 200 milliseconds in Figure 4.7. With database service demand time (*D*) is 200 microseconds and the think time (*Z*) is 200 milliseconds, we get *N** to be approximated 1000 requests the database can handle without any significant delays. The difference between the database graph and the controller graph can be explained by the same reasoning as given in the analysis of *VectorMap* request.

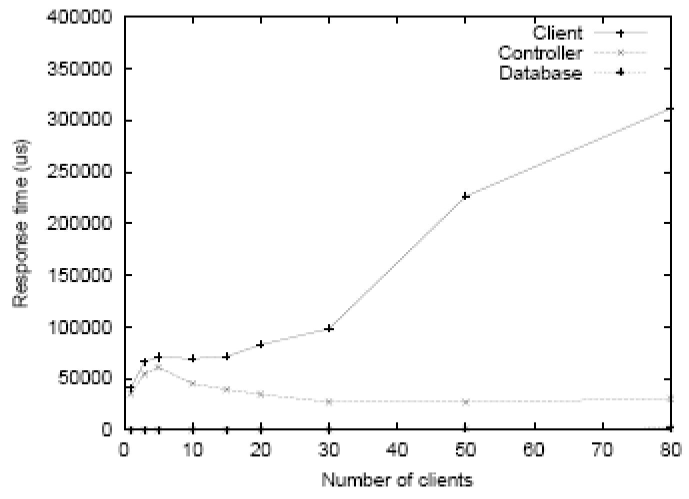


Fig.4.7. Response time of *Locate* (Z=200ms)

www.studymafia.org

CONCLUSION AND FUTURE WORK

Location-aware computing involves the automatic tailoring of information and services based on the current location of the user. We have designed and implemented Rover, a system that enables location-based services, as well as the traditional time-aware, user-aware and device-aware services. To achieve system scalability to very large client sets, Rover servers are implemented in an "action-based" concurrent software architecture that enables fine-grained application-specific scheduling of tasks. We have demonstrated feasability through implementations for both outdoor and indoor environments on multiple platforms.

Rover is currently available as a deployable system using specific technologies, both indoors and outdoors. Our final goal is to provide a completely integrated system that operates under different technologies, and allows a seamless experience of location aware computing to clients as they move through the system. With this in mind, we are continuing our work in a number of different directions. We are experimenting with a wide range of client devices, especially the ones with limited capabilities. We are also experimenting with other alternative wireless access technologies including a Bluetooth-based LAN. We are also working on the design and implementation of a multi-Rover system.

We believe that Rover Technology will greatly enhance the user experience in a large number places, including visits to museums, amusement and theme parks, shopping malls, game fields, offices and business centers. The system has been designed specifically to scale to large user populations. Therefore, we expect the benefits of this system to be higher in such large user population environments.

www.studymafia.org

REFERENCES

- www.google.com
- www.wikipedia.com
- www.studymafia.org

www.studymafia.org