Location-Aware Active Signage

by

Patrick James Nichols II

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 2004
[February 2004]

© Massachusetts Institute of Technology 2004. All rights reserved.

Author . . . .
Department of Electrical Engineering and Computer Science
January 1, 2003

Certified by . . . .
Seth Teller
Associate Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY
JUL 2 0 2004
LIBRARIES

BARKER

MITLibraries
Document Services

Room 14-0551
77 Massachusetts Avenue
Cambridge, MA 02139
Ph: 617.253.2800
Email: docs@mit.edu
http://libraries.mit.edu/docs

DISCLAIMER OF QUALITY

Due to the condition of the original material, there are unavoidable flaws in this reproduction. We have made every effort possible to provide you with the best copy available. If you are dissatisfied with this product and find it unusable, please contact Document Services as soon as possible.

Thank you.

The images contained in this document are of the best quality available.

Garyscale images only.

Location-Aware Active Signage

by

Patrick James Nichols II

Submitted to the Department of Electrical Engineering and Computer Science
on January 1, 2003, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Three-dimensional route maps, which depict a path from one location to another, can be powerful tools for visualizing and communicating directions. This thesis presents a client-server architecture for generating and displaying accurate, usable route maps between locations on MIT's campus. Two exemplary clients of this architecture – MITquest, a web based Java applet, and location-aware active signage – demonstrate the flexibility and power of this model for route generation. Additionally, we provide a framework for displaying a set of campus-wide, public events of interest to an MIT visitor, including methods for inferring events from public sources and automatically selecting events of interest.

Thesis Supervisor: Seth Teller
Title: Associate Professor of Computer Science and Engineering

Acknowledgments

This project was made possible by my parents and uncle for giving me the opportunity to study computer science at MIT, and it is to them that I dedicate my thesis. I also owe Amanda special thanks – I would not have been able to complete my studies without her love and support.

I would also like to thank Seth Teller, my thesis advisor, for his assistance, ideas, and energy. His suggestions for features and his vision for location-aware computing have been a source of inspiration in my work.

Thanks also go to Ron Wiken, the hardware guru of the 9th floor, for his patience and assistance in building the physical signs which demonstrate my work. I also benefited greatly from my partnership with Max Van Kleek, graduate student extraordinaire.

Finally, special thanks go to my colleagues in the Computer Graphics Group for their invaluable suggestions, comments, and assistance, including Jason Bell, Bryt Bradley, Michael Craig, Adel Hanna, Vitaly Kulikov, Jonathon Lee, and Peter Luka. I am especially grateful of their tolerance of my antics and idiosyncracies late into the night, and for our frequent trips to the coffee maker.

Contents

List of Figures

Chapter 1

Introduction

As computer resources become ubiquitous, inexpensive, and increasingly powerful, computing in the future will be pervasive – computer software and hardware will become an even greater part of our everyday lives, assisting in tasks both simple and complex. A common task we are faced with every day is the problem of route-finding: "How do I get from point A to point B as quickly as possible?" This thesis presents a client-server architecture for generating and displaying efficient routes between locations on MIT's campus, as well as techniques for visualizing these routes in two and three dimensions. We further provide two exemplary applications of this architecture – location-aware active signage and MITquest, a web-based Java applet.

1.1 Thesis Overview

This chapter discusses the motivations for the creation of the location-aware active signage and MITquest, as well as the contributions these applications make. This chapter then presents an overview of these applications and motivating scenarios for their respective use. The second chapter presents background material and related efforts. The third chapter focuses on the software design and architecture used in the creation of our client-server model and exemplary applications, discussing design choices and alternatives. The fourth chapter is about the user interface for both active signage and MITquest, and our methodology for route presentation. The fifth chapter touches on the physical requirements of active signs, and the subsequent design chosen. Finally, the sixth chapter presents conclusions and offers suggestions for future work.

1.2 Motivation

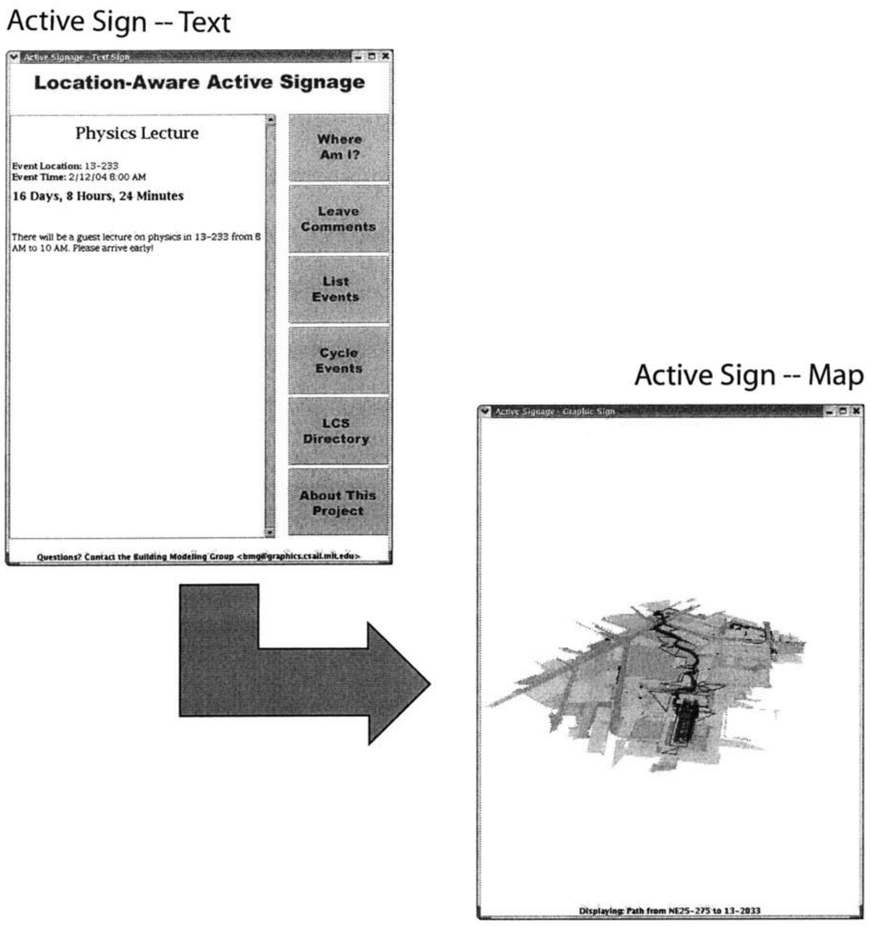
Pervasive, pose-aware applications (e.g. [8]) let users interact with the real world in new and interesting ways. As we acquire and develop more complete models of physical spaces – which can be augmented with information about furniture, people, and objects – we need more sophisticated ways of organizing and presenting these models.

This thesis is motivated primarily by a desire to provide useful methods of route-finding and map visualization for these pervasive applications. Extensive related work done by MIT's Building Modeling Group (discussed at length in Section 2.3) has produced accurate models of MIT's campus, but without methods for searching and displaying these models they are of limited use. We have thus attempted to provide techniques and software which demonstrate the substantial efforts of the BMG in a useful manner.

A second motivation for this work is to create a set of useful services for visitors to MIT, who might wish to explore MIT in person through a location-aware active sign, or via the Internet using MITquest. This thesis has endeavored to provide usable, simple applications that assist visitors in finding out about interesting events and how to find them.

Specifically, the work presented in this thesis enables the following scenarios:

1. A visitor to MIT's campus enters Lobby 7, looking for information about a public lecture being given in a few minutes in building 38. She sees a wall-mounted active sign, shown in Figure 1-1, cycling through events on campus, and notices the information about the lecture on display. She walks up



Location-Aware Active Signage

Physics Lecture

Event Location: 13-233
Event Time: 2/12/04 8:00 AM

16 Days, 8 Hours, 24 Minutes

There will be a guest lecture on physics in 13-233 from 8 AM to 10 AM. Please arrive early!

Questions? Contact the Building Modeling Group <bmg@graphics.csail.mit.edu>

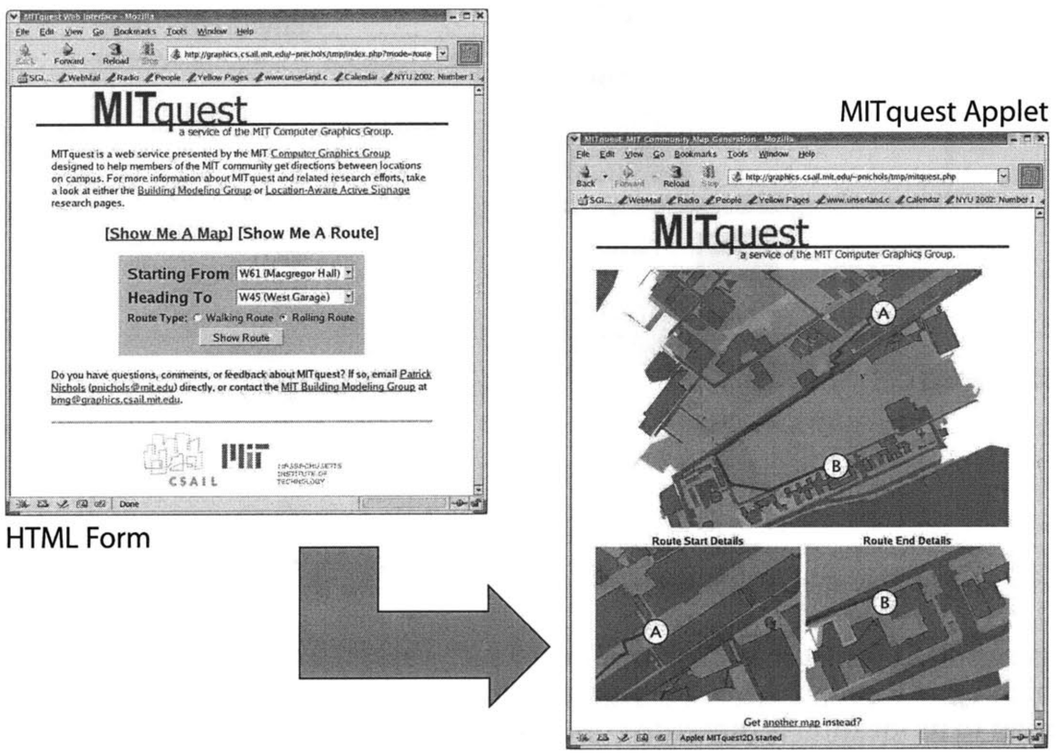
Displaying: Path from NE25-275 to 13-2033

Figure 1-1: Using Active Signage. The user is presented with community events, which cycle through the active sign display. Each time a new event is displayed, a map to its location is presented on an adjacent sign.

to the sign and is presented with a 3D route map from her current location to the lecture.

2. A student needs directions to his advisor's office. He points his web browser to the MITquest website and enters his dorm as the start location, and his advisor's building as the destination. He also notes on the web form that he would prefer a rolling route to accommodate his wheelchair. He submits his query, and is presented with a route, as seen in Figure 1-2.

3. A visiting professor has an appointment with a Course VI faculty member in her office in building NE43. The professor parks on Main Street and


MITquest Applet
HTML Form
MITquest Web Interface - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://graphics.csail.mit.edu/~pnichols/tmp/index.php?mode=route
MITquest
a service of the MIT Computer Graphics Group.

MITquest is a web service presented by the MIT Computer Graphics Group designed to help members of the MIT community get directions between locations on campus. For more information about MITquest and related research efforts, take a look at either the Building Modeling Group or Location-Aware Active Signage research pages.

[Show Me A Map] [Show Me A Route]

Starting From: W61 (Macgregor Hall)
Heading To: W45 (West Garage)
Route Type: Walking Route [✓ Rolling Route]
Show Route

Do you have questions, comments, or feedback about MITquest? If so, email Patrick Nichols (pnichols@mit.edu) directly, or contact the MIT Building Modeling Group at bmg@graphics.csail.mit.edu.

CSAIL
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Done
MITquest: MIT Community Map Generation - Mozilla
http://graphics.csail.mit.edu/~pnichols/tmp/mitquest.php
A
B
Route Start Details
Route End Details
Get another map instead?
Applet MITquest2D started

Figure 1-2: Using MITquest To Get Directions. To get directions between two buildings, the user specifies the start and end buildings in a web form, as well as the type of route. After entering the route query, the user is presented with a map showing the quickest route between locations.

enters NE43, but has forgotten the faculty member's office number in his car. He walks up to a sign in the main elevator lobby of the building, and presses the "LCS Directory" button. A list of faculty members and their respective offices pops up. He selects an office and is presented with a three-dimensional map of NE43, as well as the quickest route to the office. Before heading to his meeting, he presses the "Leave A Comment" button on the sign and offers some feedback on how to make the sign easier to use.

1.3 Contributions

This thesis makes the following contributions:

- A client-server architecture for providing a useful, abstracted route-finding and map display service
- Methods for encapsulating a two- or three-dimensional map as a Java class that can intelligently visualize routes through a standard interface
- A framework for representing campus events, for inferring them from public event calendars on the World Wide Web, and for sending them to a set of distributed clients.
- A framework for representing routes at different levels of abstraction, constrained by temporal and physical factors
- Substantial improvements to the Location-Aware API developed by Bell [2], including performance optimizations, a streamlined approach to route generation, and the addition of inter-building route-finding
- The hardware and software required to construct and deploy a set of distributed, networked active signs around MIT's campus.
- MITquest, a web-based Java applet for performing constrained route and map generation

1.4 Location Server Overview

The Location Server is a network-accessible Java RMI program which provides raw campus geometry and route-finding capabilities to client applications through a general API. The location server provides these services to active signs (see Section 3.5) and MITquest (see Section 3.6), as well as other clients.

The location server obtains campus geometry through the Building Modeling Group (BMG) pipeline, a series of applications which convert a central corpus of MIT floor plans and a campus-wide 'basemap' into a collection of spaces with known geometry and adjacencies. After this processing is complete, the location server provides access to this geometry information, and can additionally generate constrained routes between spaces and locations when requested by client applications.

1.5 Active Signage Overview

Active signs are inexpensive, portable wall-mounted computers that have been endowed with three pieces of information: knowledge of surrounding geometry, knowledge of the sign's physical location and orientation within that geometric context, and knowledge of a set of public events happening on campus. Using this information, signs intelligently select and display events to show to passers-by, and provide custom maps from their present location to the event in question. The software infrastructure supporting this project is designed to scale to hundreds of networked signs distributed across campus, all accessing a central source of geometry and event information.

Knowledge of campus geometry is furnished by a Location Server, a concept first explored by [2] and substantially expanded in this work. Both pure physical geometry (e.g. the contours of a room) and routes between spaces (e.g. the path between one room and another) are provided through a network-transparent Java RMI interface from a central Location Server to individual signs. The Location Server is discussed in greater detail in Section 3.3.

Knowledge of campus events is furnished by an Event Server, which obtains event data by crawling public MIT community websites and inferring events directly. Events are broadly construed to include occurrences like lectures, presentations, and emergencies, all of which have both a time and a location. The central event server is accessed via a network interface, and distributes events as they are created to all registered active signs. More information on the Event Server and related components can be found in Section 3.5.3.

Finally, knowledge of location is furnished either manually (it is entered into a sign by a human), or is actively acquired from the environment from an external source such as Cricket [7] or GPS. Signs that obtain their location in such a manner can perform real-time, accurate mapping – allowing a traveling sign to continuously display its location, bearing, and surroundings.

1.6 MITquest Overview

MITquest is a web-based Java applet that provides custom maps of and routes on MIT's campus, using the same data sources and algorithms found in active signs. MITquest is designed to assist campus visitors in the common task of finding their way from one place to another, with the additional ability of specifying route constraints. For example, a visitor might wish to only view routes that are wheelchair-accessible, do not require an MIT ID card, or that minimize time or path length outside of buildings. MITquest provides an interface for specifying the source and destination of routes, as well as a host of constraints upon route generation.

MITquest uses a Location Server to obtain geometry and route information, using the same abstractions and methods as active signs. Providing a web interface to the same information serves to establish the flexibility of the underlying software design powering the active signs, as well as serving a useful function for members of the MIT community. MITquest's features and design are further explored in Section 3.6.

Chapter 2

Background

This chapter explains background material relevant to this thesis, including efforts by previous researchers and related work. In particular, this chapter discusses the MIT Building Modeling Group (BMG) and the pipeline the BMG has developed to provide accurate geometry data of MIT's campus. The Location Aware API and the initial implementation of this API is then discussed, as are improvements subsequently made to the API. The basis for 2D and 3D route-finding is presented, including a brief discussion of triangulation, Dijkstra's algorithm, and iterative shortening.

2.1 Introduction and Related Work

This thesis links substantial work which has been done on two fronts: the construction of interesting pervasive applications, such as intelligent electronic kiosks, and efforts to create highly-usable route-maps.

2.1.1 The Ki/o Kiosk Platform

Both the physical design and the software of active signs has been influenced by the work of Max Van Kleek [10] in his development of Ki/o, an interactive electronic kiosk. Ki/o shares design goals that are similar to active signage (such as educating visitors about events and locations), and correspondingly the lessons learned in the development of Ki/o have not been lost in our design of active signs.

2.1.2 LineDrive and Usable Maps

Conveying information through route maps is a challenge. The visual cues people find most effective in route maps vary from task to task, and the most usable kinds of route maps are rarely those with the highest physical fidelity. The work done by Maneesh Agrawala [1] in developing systems like LineDrive has inspired much of the route display work presented in this thesis. The observation that routes and physical geometry can be effectively conveyed with less, rather than more, information is a point we have considered and tried to incorporate in our presentation of routes in both active signage and MITquest.

2.2 Primary Sources of Physical Campus Data

The methods for route-finding we present would be of little value without a large, complex data set to test them on. Obtaining a physically accurate, meaningful model of MIT's campus, however, is a challenging proposition. Although we had the option of creating our own models of MIT's campus using standard tools such as AutoCAD, the task of creating physically accurate representations of over 10,000 rooms in over 150 buildings was intractable. Rather, we have based our efforts on the work of others (namely the BMG pipeline) to extract meaningful 3D models of campus from existing data sources.

These data sources are primarily a set of CAD DXF floor plans maintained by MIT's Department of Facilities. There are two such sources:

1. Individual *building floor plans* with known conventions, uniquely named spaces, and implied adjacency information. These floor plans are publicly

available on the web at http://floorplans.mit.edu and are regularly updated as MIT buildings change over time.

2. A *campus basemap* specifying global building position and orientation, as well as physical campus layout, including the location of roads, sidewalks, ramps, and other physical infrastructure.

In practice, extracting a meaningful model of campus requires substantial effort due to inconsistency in the representation and layout of buildings in these CAD files, as well as errors in the files themselves. Methods for extracting this information have been implemented as a non-interactive pipeline of programs, explained in greater length in Section 2.3.

2.3 The Building Modeling Group

The Building Modeling Group (depicted in Figures 2-1 and 2-2) is a research effort within the MIT Computer Graphics Group to extract 3D models of MIT's campus from a vast corpus of CAD DXF files maintained by MIT's Department of Facilities. The BMG pipeline is a set of non-interactive applications which sequentially extract information from input DXF files, process the extracted information to remove errors and inconsistencies, and output clean, well-formed data in known file formats.

Thus, the BMG pipeline serves as a mechanism for acquiring detailed, physically-accurate geometry of MIT's campus. The pipeline makes use of both a set of DXF building floor plans and a global campus basemap, which specifies building position and general exterior campus geometry.

Each building's geometry is contained in a set of CAD files with a local coordinate system. We process the basemap to determine where buildings should be located and aligned using the "state plane" coordinate system, a standard established by the commonwealth of Massachusetts.

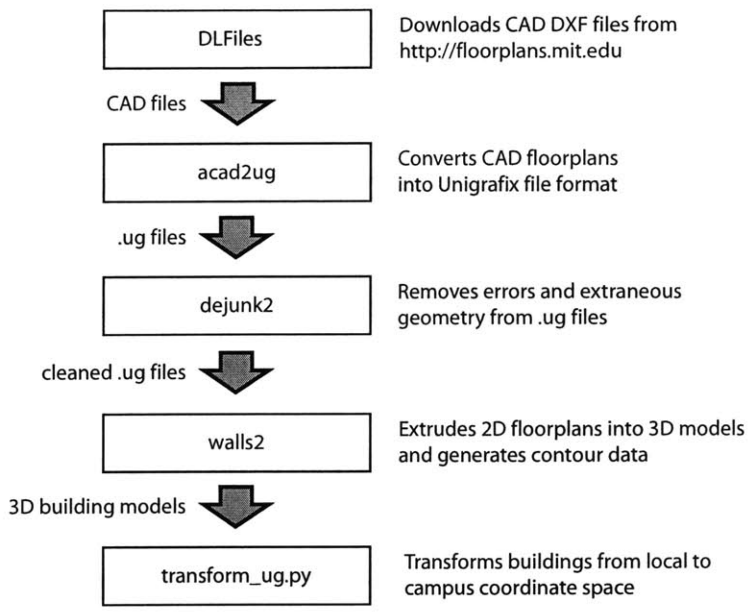

DLFiles
Downloads CAD DXF files from http://floorplans.mit.edu
CAD files
acad2ug
Converts CAD floorplans into Unigrafix file format
.ug files
dejunk2
Removes errors and extraneous geometry from .ug files
cleaned .ug files
walls2
Extrudes 2D floorplans into 3D models and generates contour data
3D building models
transform_ug.py
Transforms buildings from local to campus coordinate space

Figure 2-1: Stages of the Building Modeling Group (BMG) Pipeline. A set of non-interactive programs download CAD DXF files from floorplans.mit.edu, remove errors, extrude the floor plans into 3D, and finally transform each space from local to global space.

Additionally, we extract information about outdoor spaces (such as streets, sidewalks, and outdoor ramps) so that inter-building routes can be generated. The campus basemap has a limited amount of information, as it specifies boundaries between different kinds of campus space types, such as building/sidewalk and sidewalk/street boundaries. Like the campus floor plan files, the basemap is replete with errors and inconsistencies. We have developed methods for dividing the basemap into "patches" and coloring them as different space types using a randomized approach [4].

The final product of the pipeline is a set of text files which are read by a Location Server and represented as a set of Java objects. The actual representation of campus geometry and adjacency is discussed in greater length in Section 2.4.

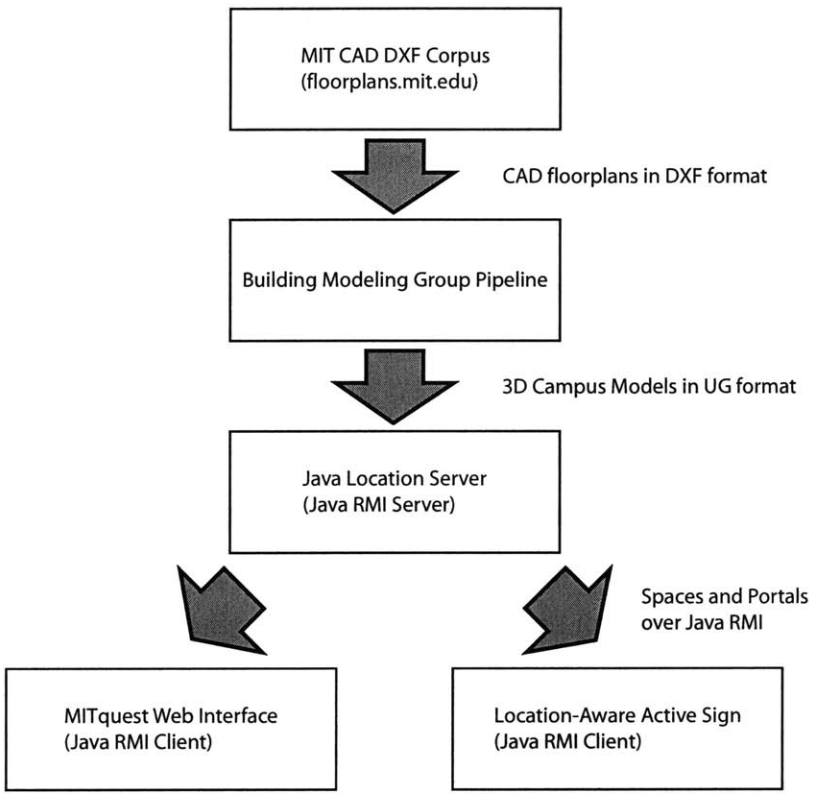


Figure 2-2: The BMG Pipeline and Location Aware API. The BMG pipeline converts MIT's corpus of CAD DXF files into a set of text files which are read by a location server, which furnishes spaces and portals over Java RMI.

2.4 Location Aware API

Our representation of the physical world emphasizes geometry and adjacency using Spaces and Portals. [9] Specifically, these objects are:

- Spaces - Closed locations bounded by an ordered polyline of 3D points. Each space has a globally unique name, following the convention: BUILDING#FLOOR#ROOM#TYPE. For example, MIT room NE43-253 would be named mit_NE43#2#253#OFF. The Space naming convention is explored in greater depth in Appendix C.1.

- Portals - Named entities representing an adjacency relationship between two Spaces. Portals are directional, indicating that one can physically move from Space S_1 to an adjacent Space S_2, though not necessarily from S_2 to S_1 This convention captures the physical fact that security doors, for example, only allow exit and not reentry. Like Spaces, Portals have a globally unique name and an associated type. Portals are physically represented by a quadrilateral, which indicates the shape of the door or elevator shaft to which the Portal corresponds. The Portal format and naming convention is further discussed in Appendix C.2.

The details of the location server file format and API are discussed in greater detail in Appendix C. This model of the world – as a set of Spaces connected by Portals – is visually depicted in Figure 2-3. The Location Aware API represents Spaces and Portals as Java objects of the same name, and makes these objects – as well as the route-finding methods discussed in Section 2.5 – available over a network interface via a Java Location Server.

Geometry and adjacency data is provided to the location server by the BMG pipeline, with two text files per building – one providing room geometry, and the other portal and adjacency information.

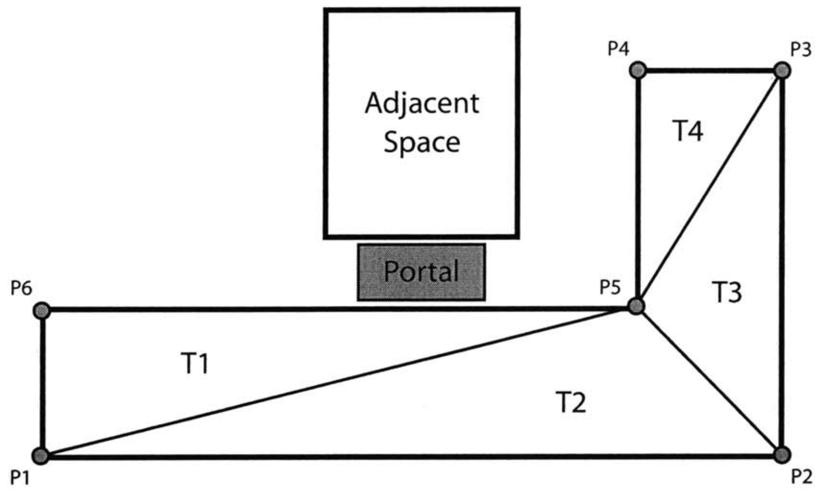


Figure 2-3: A Sample Space and Portal. Spaces are defined by a polyline of points $P1...P6$, which define the outline of the space, and the set of triangles $T1...T4$ which correspond to the space's Constrained Delaunay Triangulation.[5] Portals are defined by a set of four points corresponding to their 2D footprint, and the two spaces they connect.

2.5 Route-finding Algorithms

We have subdivided the problem of generating indoor routes into two distinct sub-problems: finding the path between two arbitrary positions within a space, and finding the collection of spaces which optimally connect the source and destination spaces. The problem of general route-finding is considered in Section 2.5.1, while our basic approach to finding paths within spaces in the focus of 2.5.2.

The fundamental algorithm we have selected for graph searching is Dijkstra's algorithm, which can be applied to route-finding over a collection of Spaces and Portals. This algorithm is applied to two searching tasks: finding a collection of Spaces, connected by Portals, which link a named source and destination; and finding the actual path through each Space in a known route. The first search is done by considering Spaces to be "nodes" and Portals to be "edges" for the purposes of graph construction and traversal. Generating paths between Spaces is as simple as applying standard graph searching techniques to a graph consisting of Space nodes linked by Portal edges. The second search, the subject of Section 2.5.2, is done by breaking each space into a set of adjacent triangles and considering each triangle as a node with known adjacencies.

2.5.1 Dijkstra's Algorithm

Dijkstra's algorithm obtains the shortest weigh path from a source node s to every other node in a graph G, which consists of weighed edges connecting nodes. It uses an initialization method, INITIALIZE-SINGLE-SOURCE, which maps an initial distance of infinity to each node, except for the source node, which has an initial distance of 0 from itself. This method initializes d, the lowest-cost distance estimate from the source to any other node in the graph, and π, the previous node. When the algorithm is complete, the shortest path from a source node s to any other node n in the graph is obtained by recursively querying $\pi[n]$ until s is obtained.

```
INITIALIZE-SINGLE-SOURCE(G, S)
1  for each vertex $v \in G$
2        do $d[v] \leftarrow \infty$
3           $\pi[d] \leftarrow \emptyset$
4  $d[s] \leftarrow 0$
```

The algorithm updates the distance estimate d with the RELAX method, which takes two nodes u and v, and a weight function w. If the distance estimate to v from the source is reduced by first travelling through u, the shorter distance value is saved and $\pi[v]$ is updated accordingly.

```
RELAX(U, V, W)
1  if $d[v] > d[u] + w(u, v)$
2     then $d[v] \leftarrow d[u] + w(u, v)$
3           $\pi[v] \leftarrow u$
```

Finally, DIJKSTRA maintains a priority queue Q containing nodes to explore and a set S of spaces for which the minimum cost path has been computed. The queue is ordered by the distance estimate of each node from the initial source, and supports an EXTRACT-MIN method which returns the least distant node. The algorithm repeatedly relaxes edges between neighboring nodes, buiding a set of least-cost paths from the start node s to all other connected nodes.

```
DIJKSTRA(G, W, S)
1  INITIALIZE-SINGLE-SOURCE(G, S)
2  $S \leftarrow \emptyset$
3  $Q \leftarrow V[G]$
4  while $Q \neq \emptyset$
5      do $u \leftarrow$ EXTRACT-MIN(Q)
6         $S \leftarrow S \cup \{u\}$
7         for each vertex $v$ adjacent to $u$
8             do RELAX(U, V, W)
```

2.5.2 Space Triangulation

To generate route maps we must determine the physical path taken through a discovered set of spaces for several reasons. First, it is not always obvious when moving between spaces which portal should be used – for example, a long corridor with many doors could afford a traveler many ways to move from one space to another, thus making a route which consisted only of spaces ambiguous. Second, when visualizing routes people often prefer a physical trail to follow through twisty, winding corridors. A street map which merely displayed adjacencies without marking the actual physical path taken conveys much less information than its standard counterpart to the confused traveler.

Additional work must be done to determine the physical path taken through a collection of spaces. For convex spaces, a simple straight line from an entry portal to an exit portal represents the shortest possible path through the space.

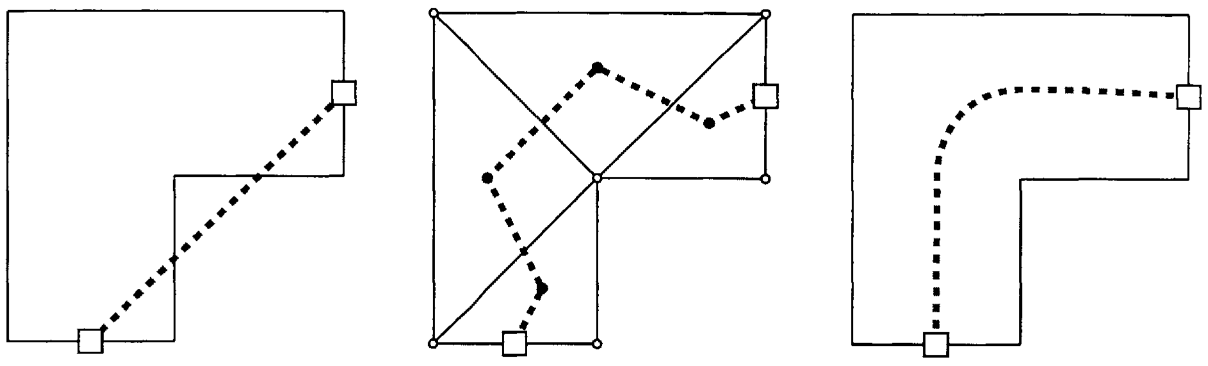


Figure 2-4: Route-finding in a 2D space with and without triangulation. In the first example, a straight line in the space results in an illegal path. In the second example, searching through the triangulated space yields a valid, if twisty path. This can be compared to a more natural path in the third example.

For concave spaces, such as the one shown in Figure 2-4, a straight line can result in an illegal path.

A solution to this problem is to triangulate concave spaces and to thus subdivide each space into a set of adjacent triangles, which are conveniently convex shapes.[5] We can then consider the set of triangles as nodes in a graph with known adjacency relationships and use the techniques mentioned in Section 2.5 to find the shortest path inside a space. Once a set of triangles is obtained, a valid path can be constructed by connecting the centroids of each triangle in the path, via the midpoint of the face shared by each pair of connected triangles.

It is important to note that this method produces valid, if not optimal, routes through spaces. Figure 2-4 shows three paths – an illegal straight line connecting the entry and exit portals for an L-shaped space, a valid path found using triangulation, and finally a more "natural" curved path. The second and third paths can be contrasted; while the triangulated path doesn't clip any of the space's boundaries, it also looks unnatural compared to the kind of path a person would likely draw when producing a map by hand. A major challenge we examine in this work is what kinds of routes look realistic, and what essential features paths should contain to compactly express the most desirable path between locations. These considerations are more deeply explored in Section 4.

Chapter 3

Software Architecture

This chapter discusses the software architecture and design choices used to implement location-aware active signage and MITquest, and provides insight into the route-finding services offered by the location server. We present an implementation of an optimized location server and a set of generic, flexible 2D and 3D graphical components which can connect to a location server for route and map visualization. Then, we show how both active signage and MITquest have been built using this client-server architecture.

3.1 Software Architecture Overview

We have designed and implemented a client-server architecture for route-finding and display, building from the efforts of others.[2] As shown in Figure 3-1, our architecture makes use of a network-available location server sharing geometry and route data over the Java Remote Method Invocation (RMI) protocol.

This architecture is motivated by a desire to make route-finding a useful service for a wide range of target applications and devices. The computational power and memory required to generate routes is beyond the current capabilities of mobile devices like cellular phones and handheld computers, so we have correspondingly pushed the effort of generating routes from the client to the server, a change from the model proposed by Jason Bell. [2] Further, the implementation effort required for a client application to make use of the services offered by a location server is greatly reduced using our model, making it easy to rapidly develop applications like MITquest.

Our goal is to enable route-finding over the entire MIT campus, which consists of over 10,000 rooms distributed across 150 buildings, with routes returned to client applications in a few seconds. Correspondingly, we have implemented a number of performance optimizations in the Location Server, discussed in Section 3.3.4, in order to enable searching across such a large data set as quickly as possible. We have further designed our event inference and distribution system, the subject of Section 3.5.3, to distribute hundreds of events to distributed networks of dozens of signs spread throughout MIT's campus.

3.2 Fundamental Data Structures

We make use of several new data structures, including:

- *Portals* – Physical connections (such as doors, elevator shafts, and stairwells) between Spaces. Each Portal has pointers to exactly two Spaces, representing one direction of the physical connection between the two. Portals also have a type (e.g. stair, elevator, door, window, ramp, etc) and a quadrilateral representing its physical "footprint" in the xy plane.

- *Spaces* – Individual rooms on the MIT campus. Each Space contains an ordered, closed polyline defining its outermost boundary, the CDT triangulation of the space, a list of the Portals exiting the Space, and pre-computed paths between the each pair of Portals in the Space. In addition to representing numbered rooms in buildings, Spaces are also used to represent sections of the campus basemap to enable route-finding between buildings.

- *Routes* – Constrained routes between two Spaces. Routes can have constraints represented by the Route type, such as "rolling" routes, which can-not use stairs. Routes also have a initial and destination space, as well as the set of connected Spaces, Portals, and physical 3D positions which best connect the source and destination spaces following constraints on the Route. The Route data structure, and its use, is explained in greater detail in Section 3.3.3.

- *Events* – Detailed descriptions of events on the MIT campus. Event objects contain the name of the Space where the event will occur, as well as a type (such as a lecture or emergency), a start time, a title, and a detailed description. Events are inferred from public MIT community event calender (see Section 3.5.3 for more details), and are distributed by a central event server to a network of distributed active signs.

- *EventQueues* – Data structures that intelligently queue and deliver Events to client applications. Each active signage client maintains a collection of active Events, but must decide which Events to display based on the event type, its physical proximity, and the current time. EventQueues, the subject of Section 3.5.4, place events into different buckets and use a weighted randomized function to select appropriate Events to display.

3.3 Location Server

We have chosen to provide a general route-finding and geometry service through a location server, whose methods are discussed in Section 3.3.1, as a Java RMI application running on a remote server. The location server reads in a set of text files provided by the BMG pipeline and instantiates Space and Portal objects corresponding to the physical rooms, doors, corridors, etc. found on MIT's campus. After loading and pre-processing this data, the location server publishes itself on the network and can be accessed over Java RMI by client applications.

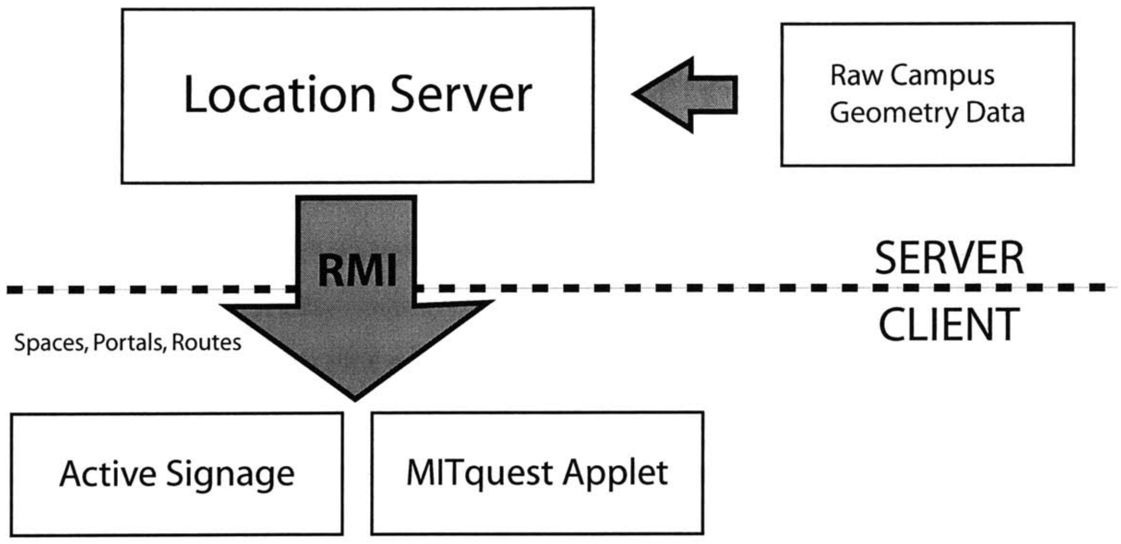


Figure 3-1: Location Server Overview. A networked location server with local access to raw geometry and adjacency information serves this data, as well as requested routes, to client applications over Java RMI.

```
public interface LocationServer {
  public Space getSpace(Position position);
  public Space getSpace(String name);
  public Portal getPortal(Position position);
  public Portal getPortal(String name);
  public Route getRoute(Route route);
}
```

Figure 3-2: LocationServer Interface

3.3.1 LocationServer Interface

The location server interface (shown in Figure 3-2) is designed to present the location server as a geometry discovery resource which clients can query for distinct spaces, portals, and routes. The model we present is that route generation is done by a powerful server rather than a client, but this is not a strict requirement, as all of the methods presented in this interface let a client implement its own algorithms for route-finding in lieu of predefined server methods.

We envision clients querying the location server an initial space, using either a known space name or a physical position obtained from a service like Cricket.

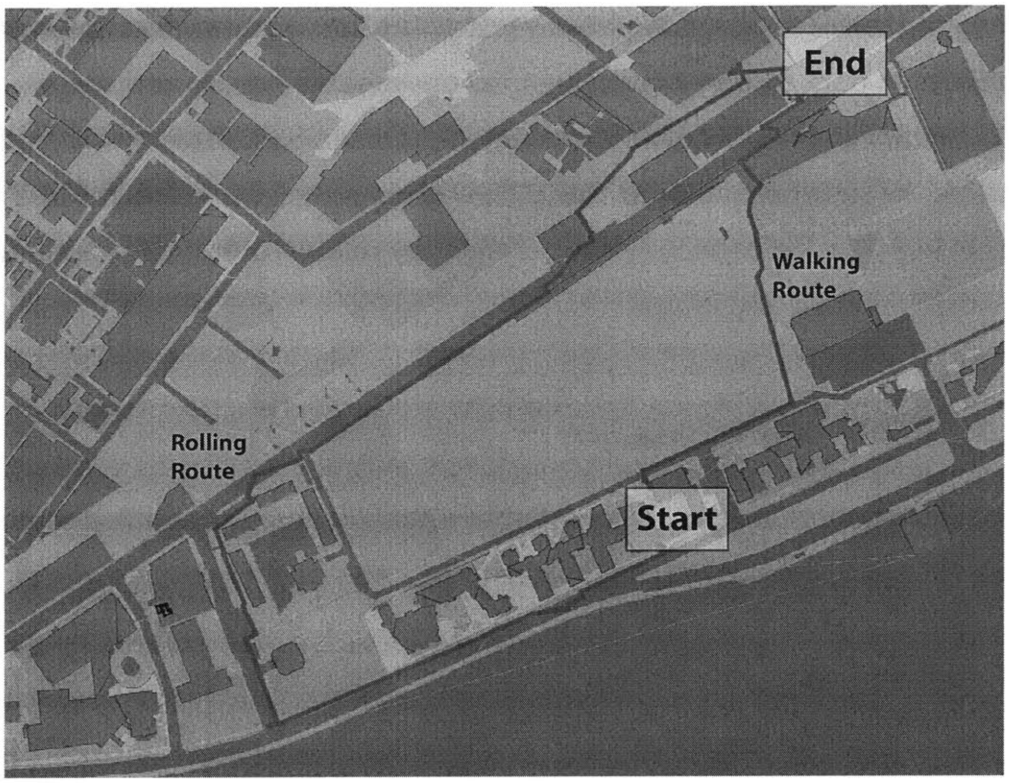


Figure 3-3: Constrained Inter-building Route Generation. We can constrain routes by filtering the relaxation step in Dijkstra's algorithm to accommodate different route types. In this example, the walking route (right) can cut through a grass field to reach the destination more quickly than the rolling route (left), which is confined to sidewalks and roads.

Then, a client can build a local map by querying the location server for spaces adjacent to its current location, and thus use a breadth first search (BFS) to discover its surrounding geometry.

3.3.2 Route-finding Algorithms

Route-finding is implemented in location server using Dijkstra's algorithm (discussed in Section 2.5), but can be reimplemented in a client application, as mentioned in Section 3.3.1. Route constraints are represented as a field in a Route object, and these constraints are used to alter path weights (Section 2.5.1, Line **??**) or selectively relax the scene graph (Section 2.5.1, Line **??**).

We implement route constraints by restricting the relaxation step in Dijkstra's algorithm, and by adding large weights to undesirable portals and spaces. Figure 3-3 shows an example of an inter-building, constrained route, with a walking route juxtaposed with a rolling route. The walking route cuts across the MIT athletics field, and is consequently shorter than the rolling route, which is confined to roads, sidewalks, and ramps.

3.3.3 Route Representation

We consider routes at many levels of abstraction. Figure 3-4 shows a route between two spaces, which we could describe in a number of ways, including:

- As some known path defined by its source and destination
- As a collection of Spaces traversed from start to finish
- As a collection of Portals passed through while moving between spaces
- As a collection of ordered, connected physical points in a polyline path

With the exception of the first item in the list above, any one of these descriptions of a route could provide sufficient information for a person to follow the route. For a simple path, knowing which Spaces (rooms) to pass through might be sufficient. For a complex path, some combination of the data above is required to easily traverse the route.

We have explicitly chosen to represent routes as rich objects containing all of the above information so that as much information as possible about a path is preserved. Our architecture delegates the task of route generation to a central location server, and the task of route display to a client application. This method of route representation lets different kinds of client applications appropriately visualize routes based on need and desired functionality. Section 3.4 discusses the use of the Java Route object in the task of map and route visualization.

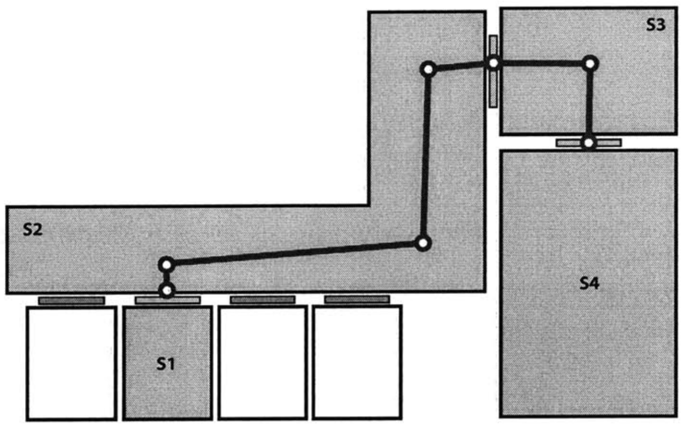


Figure 3-4: Interpretations of a Route. A route between two locations can be viewed by its source and destination locations, or as the collection of spaces, portals, or points one passes through while traveling between locations.

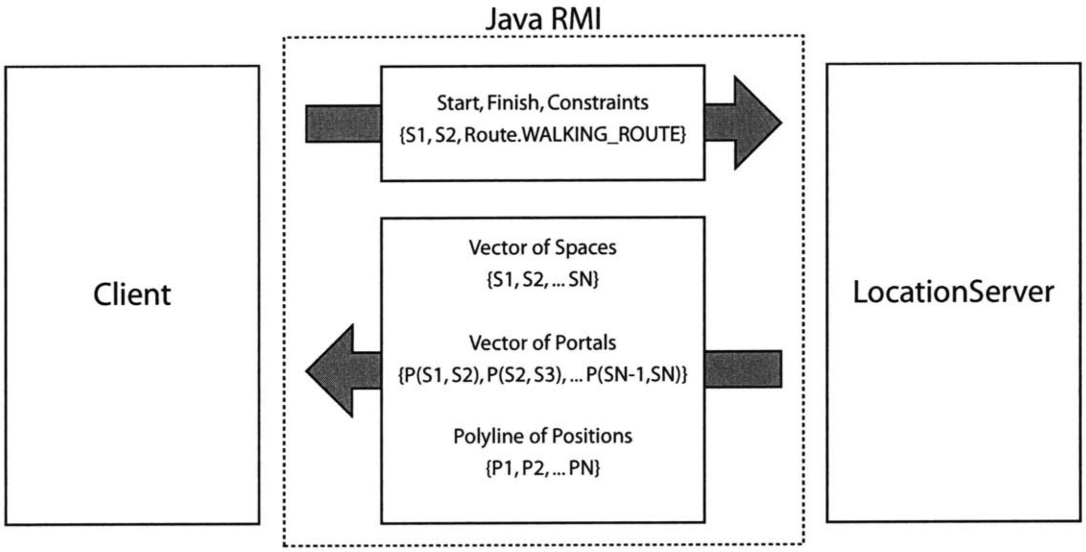


Figure 3-5: Route Passing Between Client and Server. A Route object, containing source and destination spaces and route constraints, is passed to the LocationServer over Java RMI. The Route is augmented with the Spaces, Portals, and Positions a path between the source and destination would cross before being returned to the Client.

3.3.4 LocationServer Optimizations

One requirement of the Location Server is the ability to quickly generate routes between locations. In order to facilitate this objective, we have substantially altered the Java RMI Location Server as initially developed by Bell [2] with a number of optimizations. We have consciously chosen, when possible, to move as much of the computational effort of route-finding as possible into a set-up phase of the location server. Correspondingly, although the location server has a more lengthy start up time, the speed of route searching is generally improved and this computational cost is amortized across the many routes requested by many attached clients.

Accordingly, we have implemented a number of optimizations in the location server which pre-compute useful intermediate data that speeds up route generation. For example, when Spaces are instantiated the shortest length paths between each pair of the space's portals is computed and stored. This allows for the speedy generation of routes, which can be constructed by simply appending the set of precomputed paths inside individual spaces once a high level set of portals has been obtained.

We further optimize this initialization by determining space convexity. If a given space is convex, we skip triangulation and just connect portal pairs using straight lines. For concave spaces, we use the methods discussed in Section 2.5.2 to generate paths between pairs of portals. This optimization is show in Figure 3-6, with the pre-computed paths for a convex and concave space shown.

A final optimization added to the location server is route memoization – after the location server is queried for a route, it is cached and quickly returned if queried again. Although there are an extremely large number of routes that could be potentially cached (for N spaces there are at most N^2 pairs of spaces for routes to connect – and for MIT's 10,000 spaces there are 10,000,000,000 such routes), even a relatively small cache is extremely useful. This is because the set of routes requested is likely to be a very small subset of the set of all possible routes, which

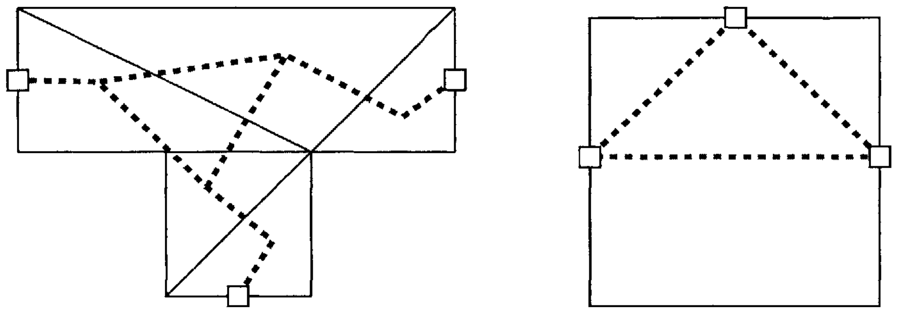


Figure 3-6: Portal-path Pre-computation With Convex Space Optimization. We pre-compute paths between portals for all spaces. Paths in the concave space at left are found by searching through the triangulated space, while paths in the convex space at right are simply straight lines.

is true for two reasons.

First, clients are likely to request routes from a relatively small number of public spaces. Most of the rooms on MIT's campus are private offices and low-traffic lobbies, and stationary clients – such as location-aware active signs – will only be positioned in a small number of public spaces. Second, the bulk of space destinations are likely, again, to be a small number of public spaces. There are a limited number of public locations where large events can be held, so memoizing routes to this reduced set of destination spaces further reduces the number of routes likely to be requested in practice.

These optimizations dramatically improve the average time required to generate a route on campus. On a full campus model with 20,000 Spaces and 80,000 Portals, we executed 1000 queries between randomly selected locations of the MIT campus. We found that the average query time for an unoptimized location was 301.2 seconds, while query time with all optimized loaded was 950 milliseconds. When looking up routes which have been previously computed and cached, the average query time was only 20 milliseconds. These results show that these optimizations and route memoization, in combination, substantially reduce the average time required to generate routes between locations.

```
public interface DisplayLayer {
  public DisplayLayer(LocationServer ls, Space initial);
  public boolean drawRoute(Route r);
  public boolean markSpace(Space s, String text);
  public void clearAll();
}
```

Figure 3-7: DisplayLayer Interface

3.4 A Generic Map Display Layer

Directions without a map are of little value. We have defined a high-level model for route and map visualized using the `DisplayLayer` interface, which is a client of a location server. Objects implementing the `DisplayLayer` interface are generic Java components which can be used in a wide variety of applications. Here, we present two such components, for 2D and 3D route display respectively.

3.4.1 DisplayLayer Interface

The `DisplayLayer` interface provides an abstract interface for applications which visualize routes and maps. The interface, shown in Figure 3-7, provides basic methods for creating a `DisplayLayer` object rooted in an initial space with an associated `LocationServer`, as well as for marking individual spaces and routes.

There are two canonical applications which implemented the DisplayLayer interface - `Canvas2D` and `Canvas3D`, which display two- and three-dimensional geometry respectively.

3.4.2 2D Route Display

We have implemented a two-dimensional `DisplayLayer` in the Java `Canvas2D` class, which itself extends the Java Canvas class. As shown in 3-8, `Canvas2D` can visualize 2D maps and routes, using color, line thickness, and shading to indicate geometry and route data. `Canvas2D` is initialized with a single starting space, and builds a complete map by querying its initial space for adjacent spaces, and recursively querying those spaces for further adjacencies. `Canvas2D` is used by the GraphicsSign application (see Section 3.5.2) to display two-dimensional routes.

3.4.3 3D Route Display

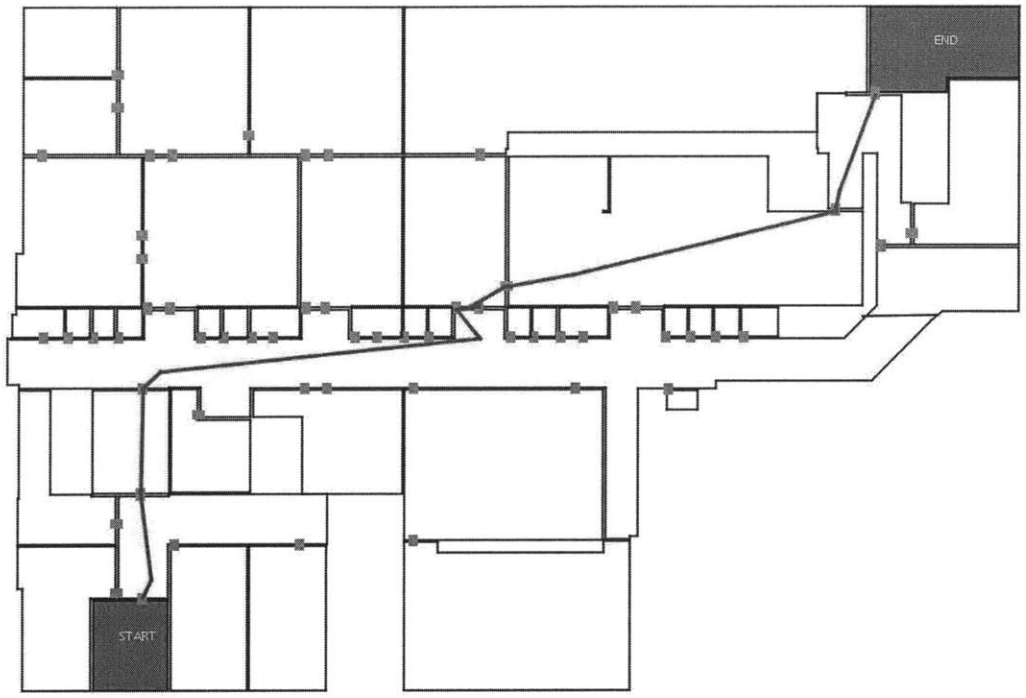
In addition to the two-dimensional implementation discussed in Section 3.4.2, we have also implements a three-dimensional `DisplayLayer` component: `Canvas3D`. `Canvas3D` builds its scene graph using the same approach as `Canvas2D`, and is similarly used by the GraphicsSign application (presented in Section 3.5.2 to display three-dimensional routes. More information about the specific techniques used to make maps visually accessible and usable is presented in Section 4.1.2.

3.5 Location-Aware Active Signage

Location-aware active signs are wall-mounted tablet PCs which cycle through a set of events on campus and selectively display maps to those events. We have implemented active signs using three applications: `TextSign`, which selects events and displays then, `GraphicsSign`, which displays maps to events as directed by a master `TextSign`, and `EventServer`, which creates and distributes events to a network of signs. We have chosen to implement active signs in pairs, with one sign running `TextSign`, and a physically adjacent sign running `GraphicsSign`. Each `GraphicsSign` is slaved to a single `TextSign`, which is in turn connected to both a location server and an event server. This architecture is visually presented in Figure 3-9.

3.5.1 TextSign

The `TextSign` application presents events and provides interactive sign functionality. The user interface of the application, shown in Figure 3-10, has two main components: a windowing area where event information is displayed, and a set



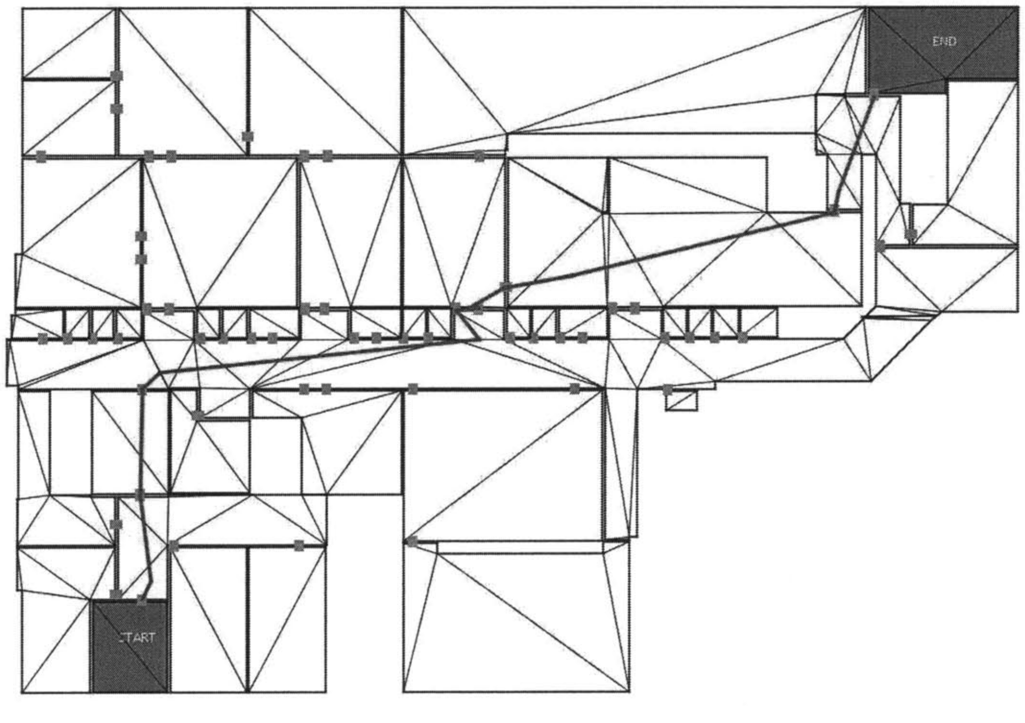


Figure 3-8: 2D Route Display. The figures above show routes and maps generated by the location server and displayed in Canvas2D. The figure at left shows an unadorned route; the figure at right shows the same route with space triangulation made visible.

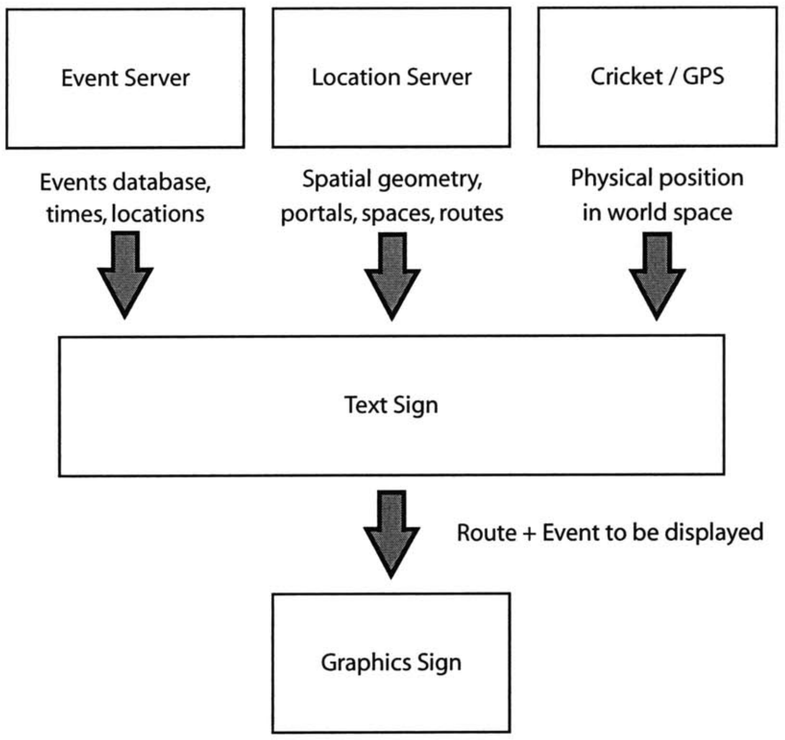

Event Server
Location Server
Cricket / GPS
Events database, times, locations
Spatial geometry, portals, spaces, routes
Physical position in world space
Text Sign
Route + Event to be displayed
Graphics Sign

Figure 3-9: Active Signage Software Overview. The TextSign component aggregates event data, campus geometry, and the sign's physical location, passing commands to a slaved GraphicsSign.

of buttons which users can click to access additional features, such as an LCS personnel directory and a pop-up dialog to leave comments.

The `TextSign` application makes a connection to an event server, and obtains a copy of the server's cached events. The `TextSign` also listens for connections from `GraphicsSigns`, to which it delegates map display. The `TextSign` periodically selects events to display, instructed attached clients to display routes from its known location to the event currently on display. The `TextSign` user interface is discussed and explored in depth in Section 4.2.2.

3.5.2 GraphicsSign

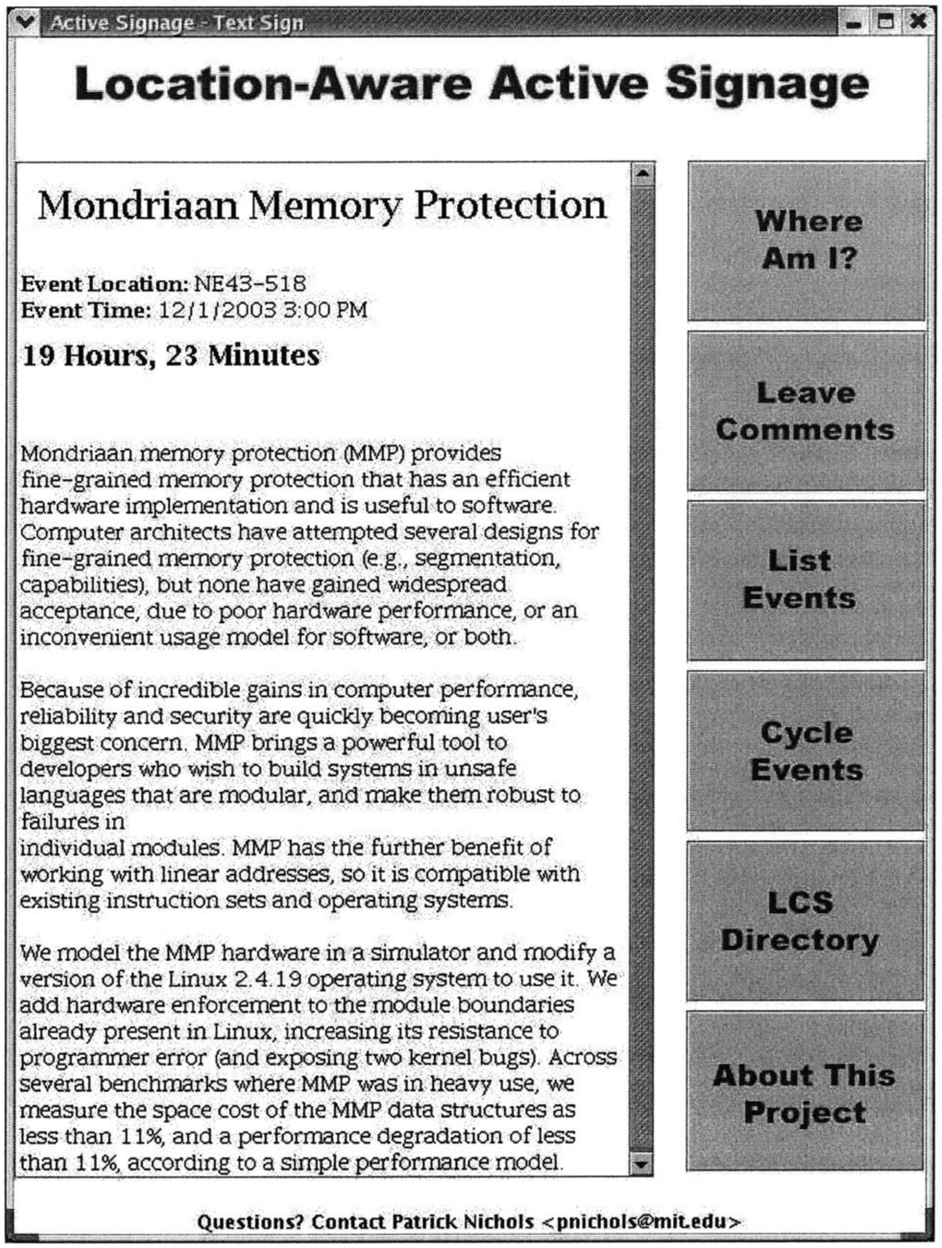
The `GraphicsSign` application presents maps as instructed by a master `TextSign`, delegating map display to either a `Canvas2D` or `Canvas3D` component embedded in the main `GraphicsSign` frame. The display itself is extremely simple, as seen in Figure 3-11.

3.5.3 EventServer

A set of campus "events" are inferred and maintained by a central event server, implemented in the Java `EventServer` class. The event server maintains a collection of Java `SignEvent` objects, which correspond to public events such as lectures. Each event has an associated time, location, type, and an optional description. Events include more than just lectures, however – we have additionally implemented an "Emergency" event type, which corresponds to emergencies like fires or chemical spills. `EventServer` is accessed by clients over a network interface, and also has an interactive graphical user interface, shown in in Figure 3-12.

Events are inferred by a central event server, which periodically probes known, public event calendars on the web. Specifically, we have implemented probes which recover elements from the MIT community events calendar[1] and the MIT

[1]Available on-line at http://events.mit.edu.



Location-Aware Active Signage

Mondriaan Memory Protection

Event Location: NE43-518
Event Time: 12/1/2003 3:00 PM

19 Hours, 23 Minutes

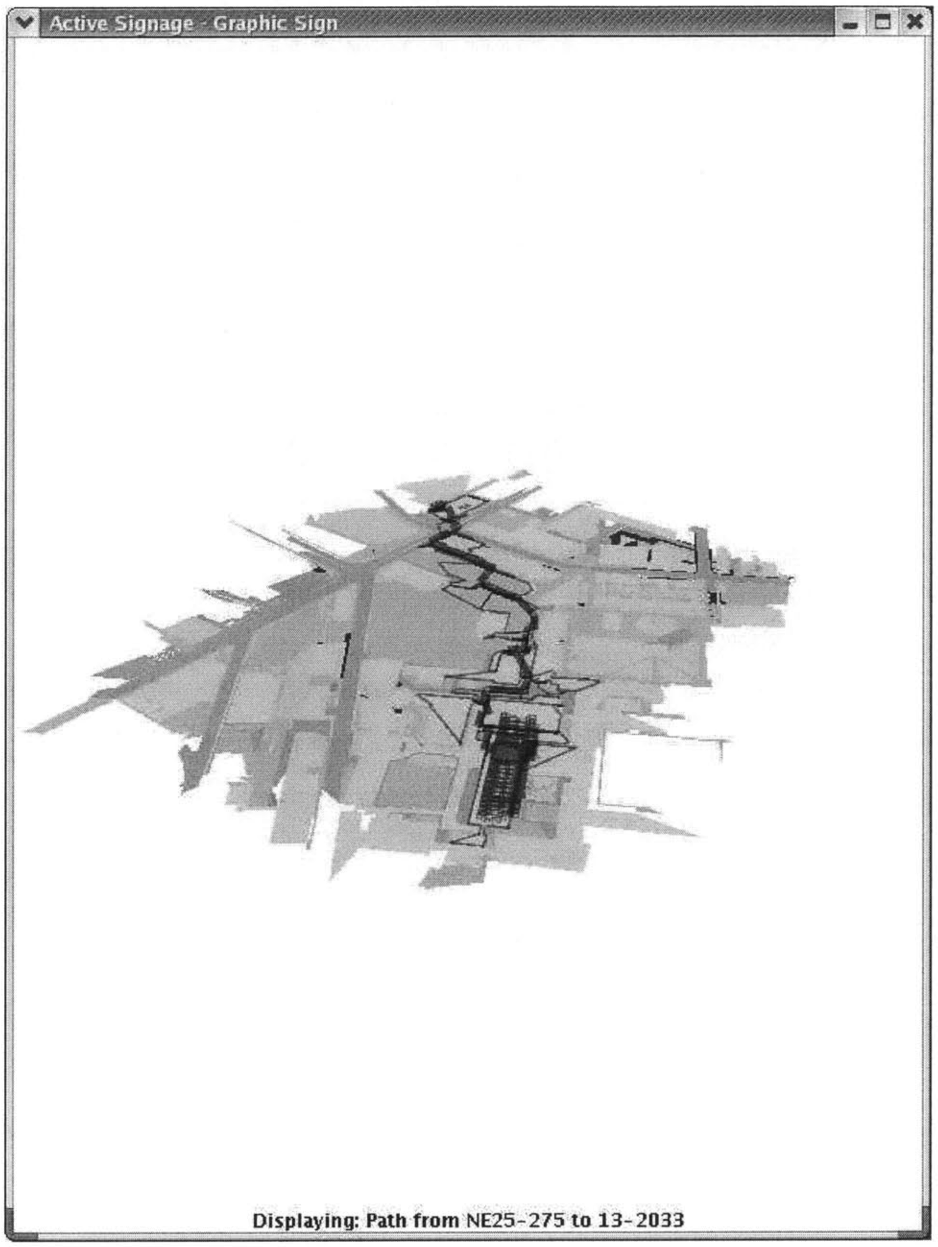
Mondriaan memory protection (MMP) provides fine-grained memory protection that has an efficient hardware implementation and is useful to software. Computer architects have attempted several designs for fine-grained memory protection (e.g., segmentation, capabilities), but none have gained widespread acceptance, due to poor hardware performance, or an inconvenient usage model for software, or both.

Because of incredible gains in computer performance, reliability and security are quickly becoming user's biggest concern. MMP brings a powerful tool to developers who wish to build systems in unsafe languages that are modular, and make them robust to failures in
individual modules. MMP has the further benefit of working with linear addresses, so it is compatible with existing instruction sets and operating systems.

We model the MMP hardware in a simulator and modify a version of the Linux 2.4.19 operating system to use it. We add hardware enforcement to the module boundaries already present in Linux, increasing its resistance to programmer error (and exposing two kernel bugs). Across several benchmarks where MMP was in heavy use, we measure the space cost of the MMP data structures as less than 11%, and a performance degradation of less than 11%, according to a simple performance model.

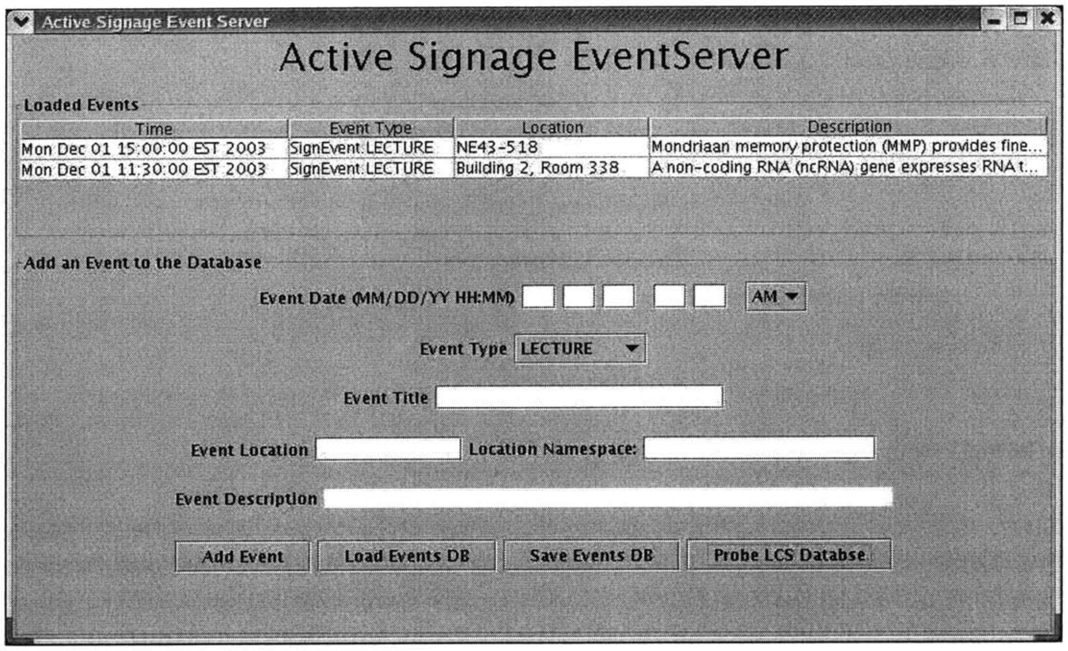
Questions? Contact Patrick Nichols <pnichols@mit.edu>

Figure 3-10: TextSign Screenshot. Event information, including the time, location, and description, are displayed in a panel on the left side of the main window. Buttons revealing different TextSign features are available in a panel on the right side of the main window.



Displaying: Path from NE25-275 to 13-2033

Figure 3-11: Graphics Screenshot. The map displayed on the GraphicsSign is controlled by a master TextSign.



Time	Event Type	Location	Description
Mon Dec 01 15:00:00 EST 2003	SignEvent.LECTURE	NE43-518	Mondriaan memory protection (MMP) provides fine...
Mon Dec 01 11:30:00 EST 2003	SignEvent.LECTURE	Building 2, Room 338	A non-coding RNA (ncRNA) gene expresses RNA t...

Figure 3-12: The EventServer GUI. Using this graphical user interface, public event sources can be probed manually, event databases saved and loaded, and events can be manually created.

Laboratory of Computer Science events page[2]. Although the general task of generating well-structured event descriptions from unformatted text can be a very challenging problem, it is thankfully one we did not face in our implementation. Rather, these two listed event sources publish events in regularly-formatted HTML and XML feeds, which are easily parsed and turned into `SignEvent` objects.

The event server thus maintains a set of events, obtained by probing and parsing public event calendars. When a client application (such as the `TextSign` application of Section 3.5.1) connects to the event server, the entire set of the event server's events are distributed as serialized Java objects to the client application. The client then decides locally which events to display, a task discussed at greater length in Section 3.5.4. The event server maintains a reference to each client, so that new events (which can be added through the interactive event server interface) can be distributed to clients. Periodically, the event server also prunes its

[2]Available on-line at http://www.lcs.mit.edu/calendar.

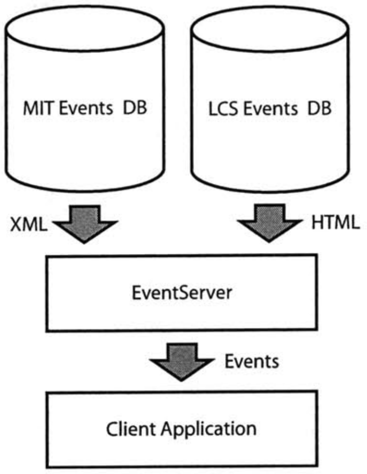


Figure 3-13: EventServer Architecture. The EventServer infers events from two sources: an XML feed from the MIT community events calendar, and HTML web pages from the LCS events calendar. The EventServer creates Event objects from these sources and distributes them to client applications.

internal set of events to discard those which have expired.

3.5.4 Event Selection

A central event server creates and maintains a central repository of events, distributing them to a network of client applications. Different clients should display different events, however, and thus make use of an event queue (implemented in the Java `EventQueue` class) to appropriately select events for display. Inferring event relevance is based on two pieces of knowledge: physical locality (how close the client physically is to the event in question) and temporal locality (how close the starting time of the event is to the current time). Obviously, clients should display events which are close and start soon more prominently than events which are distant and start in many days. We have implemented an event queue which assigns a weight to events based on this temporal and physical locality to an event. Client applications such as active signs use this event queue to manage events and extract the most relevant event for public display.

To implement this weight function, the event queue separates events into one

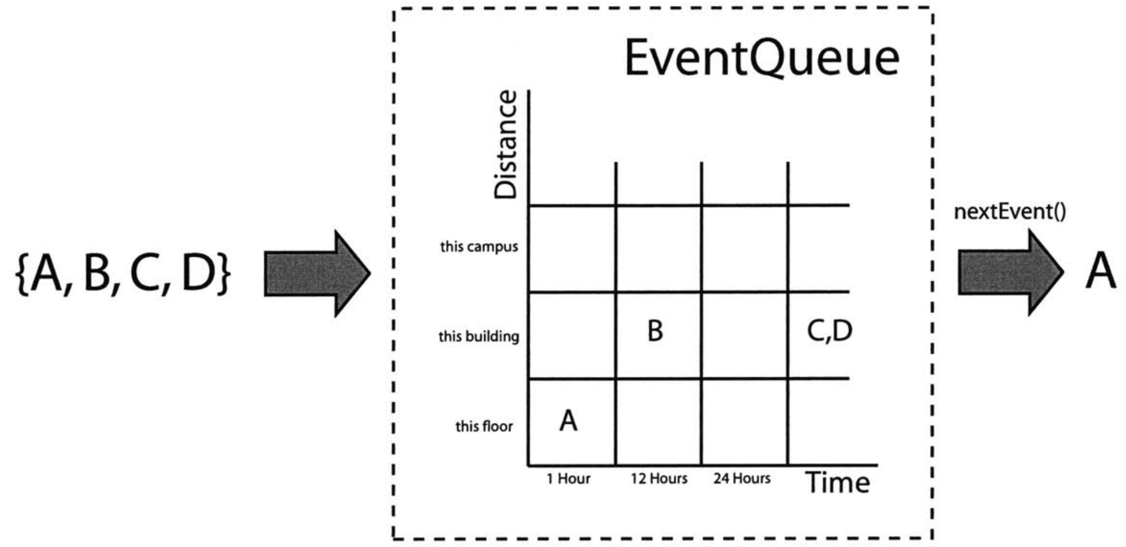


Figure 3-14: Event Queue Buckets. A set of events A, B, C, D is inserted into the event queue, with each event assigned to one of 16 buckets. Events are selected using a randomized `nextEvent()` method which is biased toward events which are close in time and space.

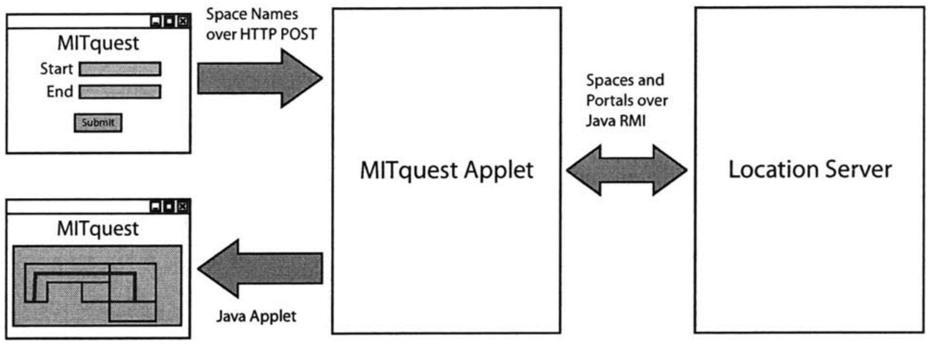


Figure 3-15: MITquest Applet Overview. Users enter route parameters, such as a start and end space, into an HTML form. The route constraints are passed over HTTP POST to the MITquest Applet, which generates the route and map with the assistance of a Location Server.

of 16 buckets, as show in in Figure 3-14. Each bucket is assigned a probability, and events are selected at random from the queue for display, with the selection function biased toward elements which are close in both time and space. This randomized selection method has the convenient property of rarely selecting duplicate methods for display, keeping an active sign from getting "stuck" on a single event for an extended period of time. At the same time, this scheme tends to present more relevant events more often.

3.6 MITquest

MITquest is implemented as a Java applet which is controlled by an HTML form embedded in a PHP script. As shown in Figure 3-15, users enter map and route requests into an HTML form, and submit the forms to the MITquest applet, which in turns connects to a location server running locally to obtain route and map data. The applet then constructs and displays the relevant image, displaying it in a client browser. Map and route display is delegated to an embedded `Canvas2D` or `Canvas3d` object.

3.7 Other Applications: GraphBuilder for Cricket

The location server and client-server approach we present in this thesis can be easily extended for other uses. One application we have developed is GraphBuilder, a location server client which is used to test Cricket self-configuration algorithms.

Cricket [7] is a location positioning system that uses a distributed network of beacons, and is designed to supplement systems like GPS for indoor use. One challenge in Cricket development is self-configuration algorithms for a large, distributed network of beacons. Ideally, once a network of beacons is seeded with position data, a newly-placed beacon should be able to determine, by "listening" to other Crickets, its own location and configure itself accordingly. As Cricket is designed for use indoors, such algorithms should be tested against real-world geometry, and GraphBuilder furnishes this geometry and test data.

GraphBuilder generates floor maps using the interface and methods presented in Section B.1, recursively querying the location server for spaces adjacent to an intial space. After generating a simple map, the application distributes Cricket beacons across the map according to a density parameter. Finally, the application links each Cricket to every other Cricket within the ideal range of the Cricket positioning system. This data is then saved to disk, where it can be viewed in Inventor format (sample output is shown in Figure 3-16) or used as input to test a Cricket self-configuration algorithm.

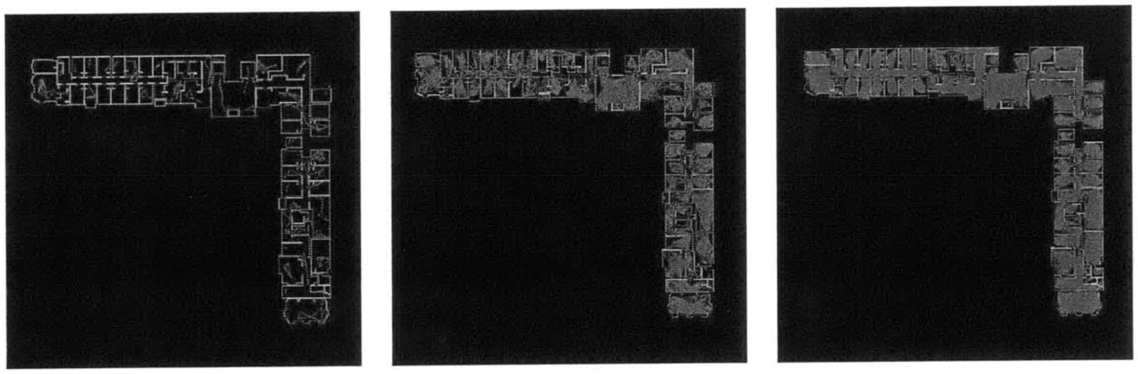


Figure 3-16: GraphBuilder Output. The GraphBuilder application distributes idealized Cricket beacons across a 2D map and then links beacons which are within range of each other. Rooms are shown in green, while Cricket-Cricket links are shown in red. A low-density Cricket distribution is show at left, a medium-density distribution in the center, and a high-density distribution at right.

Chapter 4

User Interface Design

This chapter presents our approach for making map and route display, as well as the `TextSign` application from Section 3.5.1, highly usable. We contrast the difficulties inherent in visualizing two- and three-dimensional routes, and present methods for approaching these two modes of map display.

4.1 Route Visualization

Conveying map and route information effectively is a challenging problem which has been considered in great detail for driving directions, which are often obtained through web sites like Mapquest and MapBlast. [1]. The problem of visualizing routes inside buildings, however, presents specific challenges which we have attempted to address in the 2D and 3D cases.

4.1.1 Techniques for 2D Route Display

A general problem with route display is determining the appropriate scope of the route. To show a route between two spaces, at the very least one must display the origin and destination spaces of the route, as well as the route path itself. As seen in Figure 4-1, showing too little information (path one) or too much information (path three) reduces map usability. The trick is to display an "appropriate"

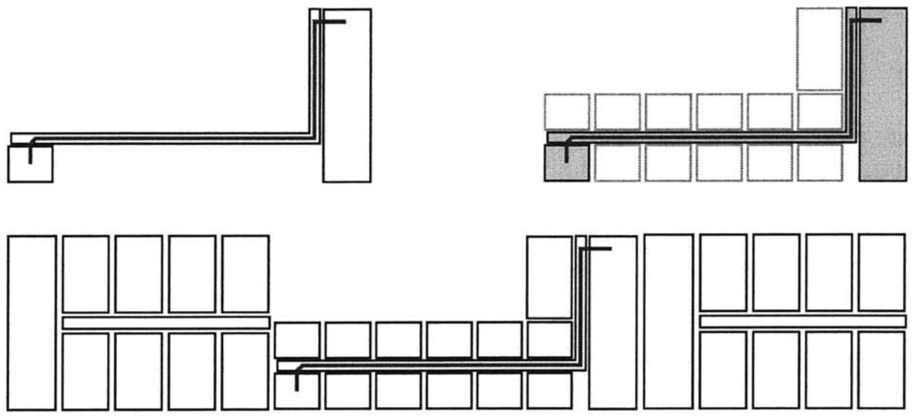


Figure 4-1: Three Paths in a Map. The first path, top left, shows very little extraneous information. The second path, top right, shows more information, but emphasizes more relevant spaces in the route. The third path, bottom, shows all local geometry.

amount of information.

We use several techniques to display two-dimensional route maps effectively. First, we use the bounding box of the route itself to clip local geometry in order to restrict the number of rooms shown on the map. Second, we use color and line thickness to emphasize important regions of a map. The contours of the path itself, for example, are shown in a brighter, thicker color than other lines. Spaces which are part of the route are filled, drawn with a thicker outline, and have distinctive colors relative to less important geometry, which is still presented to give the route context.

4.1.2 Techniques for 3D Route Display

The same challenges and techniques discussed in Section 4.1.1 apply to three-dimensional route maps, with one additional challenge: a third dimension. Traditional maps rarely make use of height information, and when they do the result is most often a projection of feature height onto (e.g. isoclines on a topology map) a plane. Occasionally illustrators use a 3/4 view of buildings with clipped walls and separated floors to show building interiors, but such views are difficult and time-consuming to manually produce. Thus, in our everyday experiences we rarely see and use three-dimensional maps.

The primary challenge in displaying a three-dimensional route map is *occlusion*: three-dimensional spaces often block each other when a building is externally viewed. For a route between two spaces in the same building, it is usually impossible to see most of the route's length - especially given that vertical adjacency structures in buildings (such as elevator shafts and staircases) are usually in the center of a building, resulting in a necessarily occluded path.

We have devised several techniques to attempt to display three-dimensional paths. First, we selectively shade lines and spaces using the techniques discussed in Section 4.1.1. Further, we selectively display spaces in a map using bounding volumes to restrict the view of campus presented. Finally, we selectively display intermediate spaces to present a most-relevant subset of a building in order to show a route. For a route between two floors of the same building, for example, we only display rooms on the floor of the source and destination space, and selectively shade rooms the route passes through. Transparency is further used to display the route path inside spaces. These techniques can be seen in Figure 4-2, where a the middle image uses the techniques described to more clearly display a route in three dimensions.

4.2 Location-Aware Active Signage User Interface

This section discusses the user interface for the `EventServer` application in Section 4.2.1 and for the `TextSign` application in Section 4.2.2.

4.2.1 EventServer User Interface

`EventServer`, the central application which acquires and distributes a collection of campus events, is controlled by the simple user interface shown in Figure 4-3. As shown in the figure, `EventServer` is controlled by a set of buttons in the

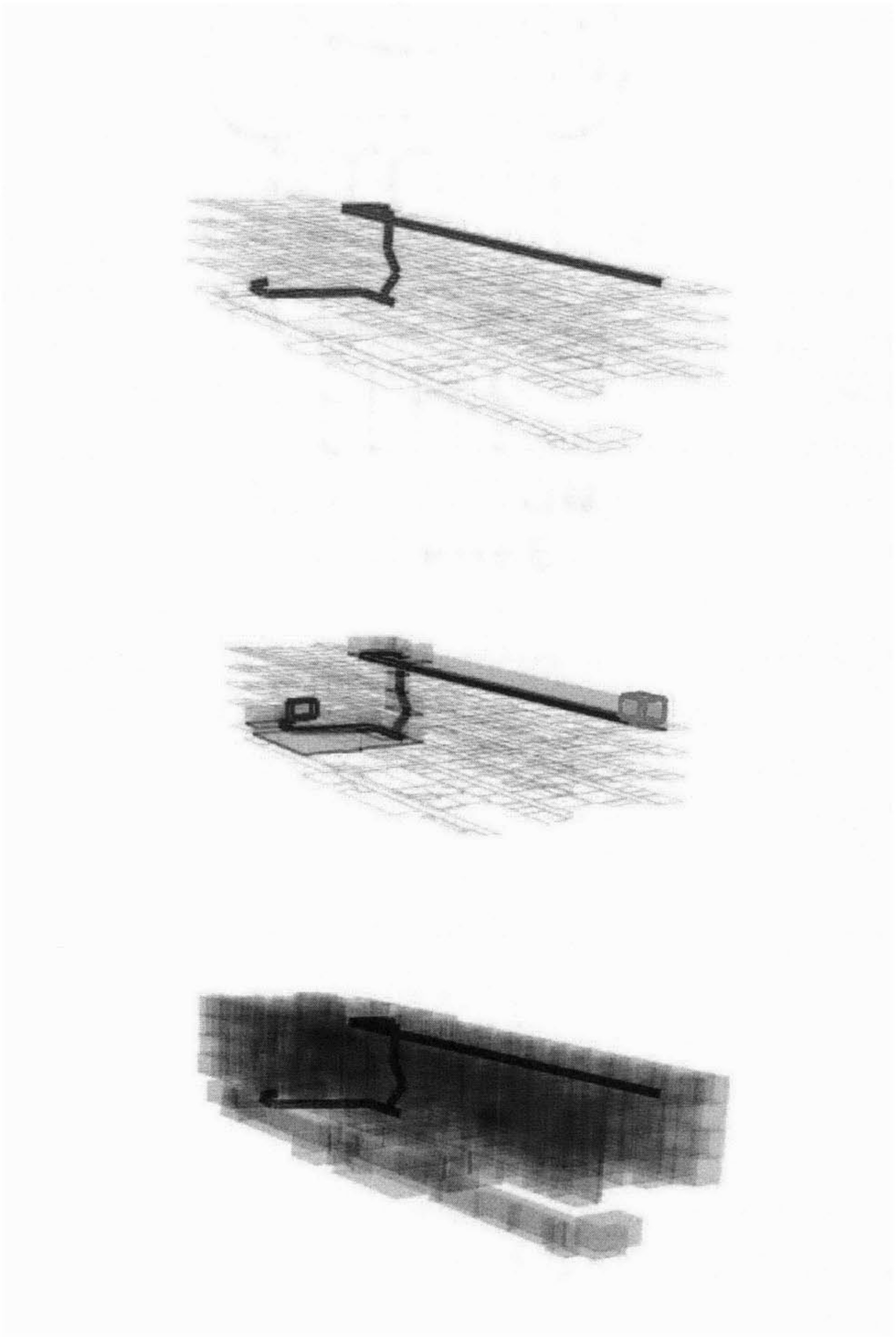
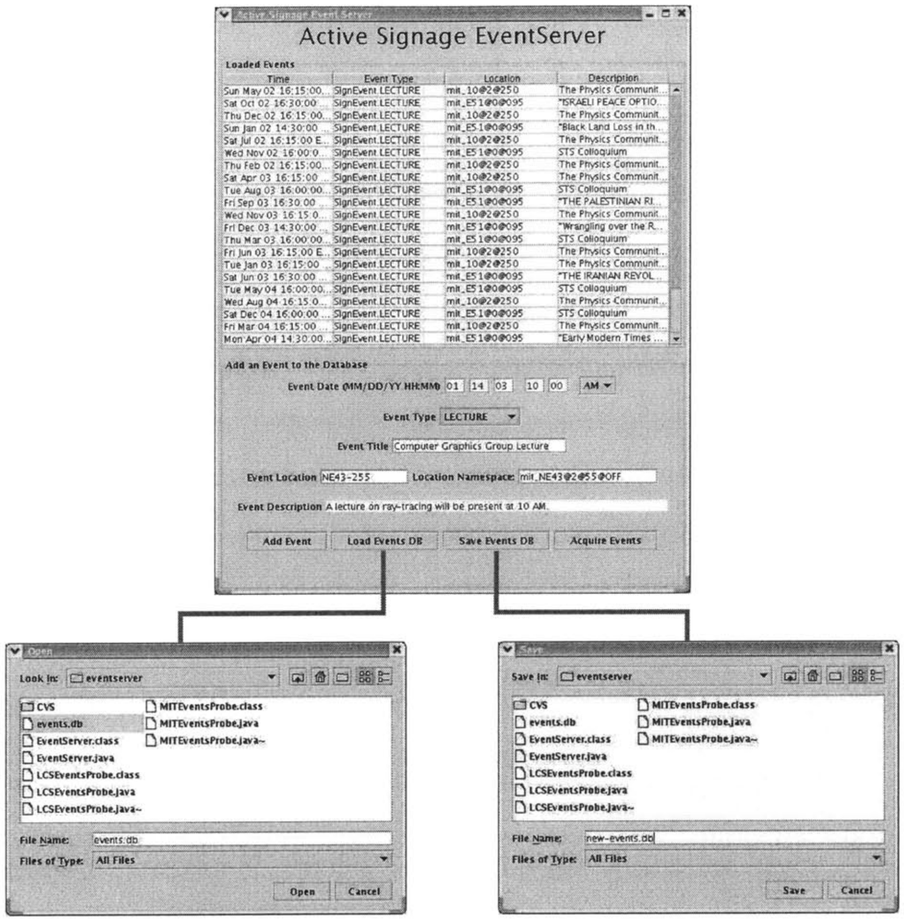


Figure 4-2: 3D Route Depiction. The map at top shows only the outlines of rooms in gray and a marked route in red. The map in the middle colors intermediate route spaces, and highlights the start space (green) and end space (blue). The map at bottom shows all spaces shaded, with the route partially occluded.


Active Signage EventServer

Loaded Events

Time	Event Type	Location	Description
Sun May 02 16:15:00...	SignEvent.LECTURE	mit_10@2@250	The Physics Communit...
Sat Oct 02 16:30:00 ...	SignEvent.LECTURE	mit_E51@0@095	"ISRAELI PEACE OPTIO...
Thu Dec 02 16:15:00...	SignEvent.LECTURE	mit_10@2@250	The Physics Communit...
Sun Jan 02 14:30:00 ...	SignEvent.LECTURE	mit_E51@0@095	"Black Land Loss in th...
Sat Jul 02 16:15:00 E...	SignEvent.LECTURE	mit_10@2@250	The Physics Communit...
Wed Nov 02 16:00:0...	SignEvent.LECTURE	mit_E51@0@095	STS Colloquium
Thu Feb 02 16:15:00...	SignEvent.LECTURE	mit_10@2@250	The Physics Communit...
Sat Apr 03 16:15:00 ...	SignEvent.LECTURE	mit_10@2@250	The Physics Communit...
Tue Aug 03 16:00:00...	SignEvent.LECTURE	mit_E51@0@095	STS Colloquium
Fri Sep 03 16:30:00 ...	SignEvent.LECTURE	mit_E51@0@095	"THE PALESTINIAN RI...
Wed Nov 03 16:15:0...	SignEvent.LECTURE	mit_10@2@250	The Physics Communit...
Fri Dec 03 14:30:00 ...	SignEvent.LECTURE	mit_E51@0@095	"Wrangling over the R...
Thu Mar 03 16:00:00...	SignEvent.LECTURE	mit_E51@0@095	STS Colloquium
Fri Jun 03 16:15:00 E...	SignEvent.LECTURE	mit_10@2@250	The Physics Communit...
Tue Jan 03 16:15:00 ...	SignEvent.LECTURE	mit_10@2@250	The Physics Communit...
Sat Jun 03 16:30:00 ...	SignEvent.LECTURE	mit_E51@0@095	"THE IRANIAN REVOL...
Tue May 04 16:00:00...	SignEvent.LECTURE	mit_E51@0@095	STS Colloquium
Wed Aug 04 16:15:0...	SignEvent.LECTURE	mit_10@2@250	The Physics Communit...
Sat Dec 04 16:00:00 ...	SignEvent.LECTURE	mit_E51@0@095	STS Colloquium
Fri Mar 04 16:15:00 ...	SignEvent.LECTURE	mit_10@2@250	The Physics Communit...
Mon Apr 04 14:30:00...	SignEvent.LECTURE	mit_E51@0@095	"Early Modern Times ...

Add an Event to the Database

Event Date (MM/DD/YY HH:MM) 01 14 03 10 00 AM

Event Type LECTURE

Event Title Computer Graphics Group Lecture

Event Location NE43-255 Location Namespace: mit_NE43@2@55@OFF

Event Description A lecture on ray-tracing will be present at 10 AM.

Open: File Name: events.db; Files of Type: All Files

Save: File Name: new-events.db; Files of Type: All Files

Figure 4-3: EventServer User Interface. Pressing the "Acquire Events" button probes the LCS events calendar and events.mit.edu for community events, which are displayed in the top half of the user interface. Pressing "Load Events DB" brings up a dialog to load an events database from file. Pressing "Save Events DB" saves the current set of events to a local file.

bottom half of the user interface, while the top half of the window is dedicated the displaying sets of events. Using these buttons, the user can enter custom events (specifying the event title, description, time, and type), direct the application to acquire events from known data sources, and load or save events to disk.

4.2.2 TextSign User Interface

Users interact with location-aware active signage primarily through the `TextSign` user interface, detailed in Figure 4-4. The interface has been designed to accommodate the touch-screen display found on tablet PCs (the physical design of active signs is considered in Section 5.1); consequently users interact with the `TextSign` application by pressing large buttons or by leaving voice messages.

Through the `TextSign` users can browse through a list of campus events, leave comments on the sign, and browse through an MIT LCS directory to obtain directions to a professor's office. If the sign is left unattended for a few minutes it goes into a "cycle" mode, cycling through its internal database of events and selectively displaying them, using the EventQueue data structure discussed in Section 3.5.4. Users can also direct the TextSign to enter cycle mode by pressing the "Cycle Events" button, shown in Figure 4-4.

4.3 MITquest User Interface

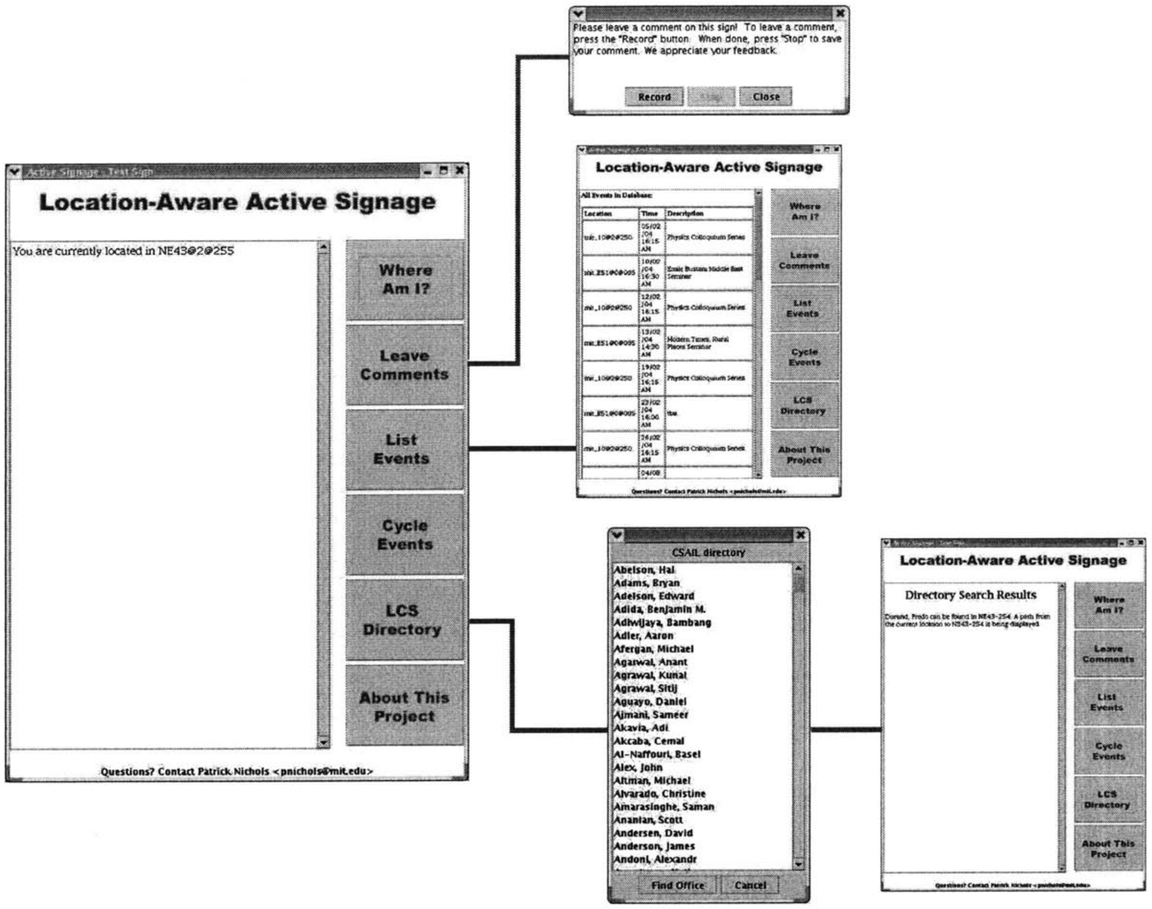
MITquest has three major components: the HTML and PHP pages used to request routes and maps, map display mode, and route display mode. Here we consider each of these user interfaces.

4.3.1 Route and Map Requests

Users request maps and routes through an HTML/PHP web form, shown in Figure 4-5. When requesting a map the user enters an MIT building and room through a simple drop down menu. When requesting a route, the user can specify the start and end locations through drop down menus, as well as the type of route (rolling or walking) required.

4.3.2 Map Display

A typical map is shown in Figure 4-6. The selected feature is displayed on a campus map with its outline inked in red, with surrounding campus geometry displayed.



You are currently located in NE43@2@255

Questions? Contact Patrick Nichols <pnichols@mit.edu>

Please leave a comment on this sign! To leave a comment, press the "Record" button. When done, press "Stop" to save your comment. We appreciate your feedback.

All Events In Database:

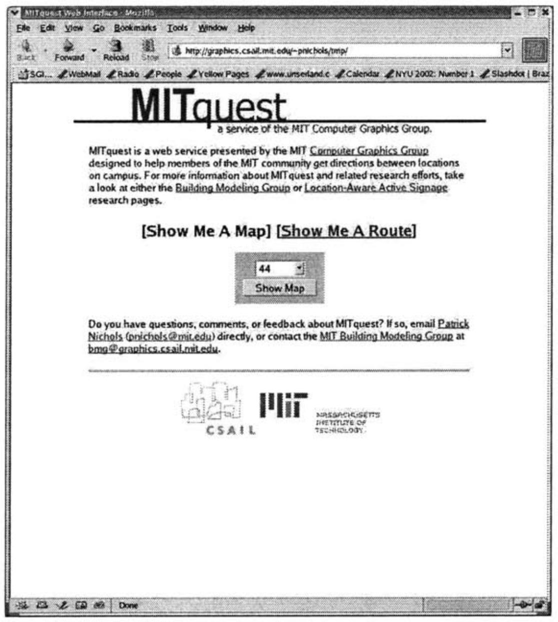
Location	Time	Description
mit_10@2@250	05/02/04 16:15 AM	Physics Colloquium Series
mit_E51@0@005	10/02/04 16:30 AM	Emile Bustani Middle East Seminar
mit_10@2@250	12/02/04 16:15 AM	Physics Colloquium Series
mit_E51@0@095	13/02/04 14:30 AM	Modern Times, Rural Places Seminar
mit_10@2@250	19/02/04 16:15 AM	Physics Colloquium Series
mit_E51@0@005	23/02/04 16:00 AM	tba
mit_10@2@250	26/02/04 16:15 AM	Physics Colloquium Series
	04/08	

Abelson, Hal
Adams, Bryan
Adelson, Edward
Adida, Benjamin M.
Adiwijaya, Bambang
Adler, Aaron
Afergan, Michael
Agarwal, Anant
Agrawal, Kunal
Agrawal, Sitij
Aguayo, Daniel
Ajmani, Sameer
Akavia, Adi
Akcaba, Cemal
Al-Naffouri, Basel
Alex, John
Altman, Michael
Alvarado, Christine
Amarasinghe, Saman
Ananian, Scott
Andersen, David
Anderson, James
Andoni, Alexandr

Directory Search Results

Durand, Fredo can be found in NE43-254. A path from the current location to NE43-254 is being displayed.

Figure 4-4: TextSign User Interface. Pressing the "Where Am I" button displays the sign's current location, and produces a map marking this location on an adjacent GraphicsSign. Pressing the "Leave Comments" button pops up a dialog which the user can use to record comments or a brief message. Pressing the "List Events" button displays a list of all the events stored in the TextSign, formatted into an HTML table. Finally, pressing the "LCS Directory" button pops up a dialog with the entire MIT Laboratory for Computer Science listed. Selected an individual name displays the person's office number, as well as producing a map from the sign's current location to the office on the adjacent GraphicsSign.

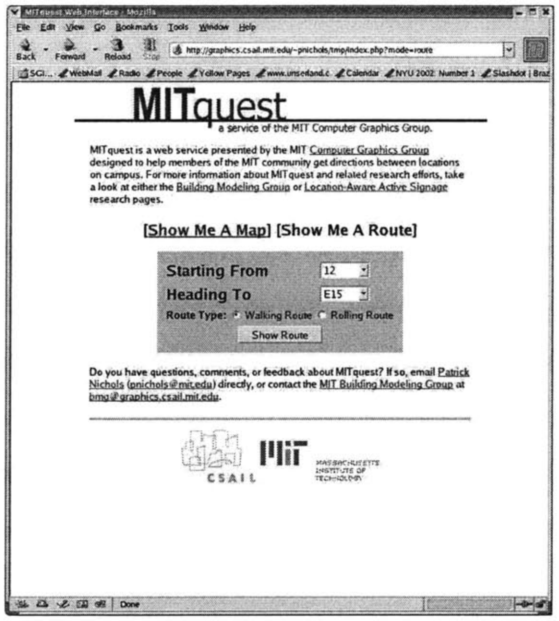

MITquest Web Interface - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://graphics.csail.mit.edu/~pnichols/tmp/
SGI... WebMail Radio People Yellow Pages www.userland.c Calendar NYU 2002: Number 1 Slashdot | Braz
44
Show Map
CSAIL
MIT
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Done

MITquest
a service of the MIT Computer Graphics Group.

MITquest is a web service presented by the MIT Computer Graphics Group designed to help members of the MIT community get directions between locations on campus. For more information about MITquest and related research efforts, take a look at either the Building Modeling Group or Location-Aware Active Signage research pages.

[Show Me A Map] [Show Me A Route]

Do you have questions, comments, or feedback about MITquest? If so, email Patrick Nichols (pnichols@mit.edu) directly, or contact the MIT Building Modeling Group at bmg@graphics.csail.mit.edu.


MITquest Web Interface - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://graphics.csail.mit.edu/~pnichols/tmp/index.php?mode=route
SGI... WebMail Radio People Yellow Pages www.userland.c Calendar NYU 2002: Number 1 Slashdot | Braz
Show Route
CSAIL
MIT
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Done

MITquest
a service of the MIT Computer Graphics Group.

MITquest is a web service presented by the MIT Computer Graphics Group designed to help members of the MIT community get directions between locations on campus. For more information about MITquest and related research efforts, take a look at either the Building Modeling Group or Location-Aware Active Signage research pages.

[Show Me A Map] [Show Me A Route]

Starting From: 12
Heading To: E15
Route Type: [✓ Walking Route] Rolling Route

Do you have questions, comments, or feedback about MITquest? If so, email Patrick Nichols (pnichols@mit.edu) directly, or contact the MIT Building Modeling Group at bmg@graphics.csail.mit.edu.

Figure 4-5: MITquest User Interface. Users can request either maps (centered on one location) shown at left, or routes (between two locations), shown at right. Map and route parameters are entered through a simple web-based form.

4.3.3 Route Display

A typical route is shown in Figure 4-7. An overview of the entire route is shown at the top of the results page, which detailed views of the start and end of the route are shown in smaller windows below. Further, the start and end spaces are labeled to increase usability. The path itself is shown as an overlaid thick red line, which contrasts with the campus basemap.

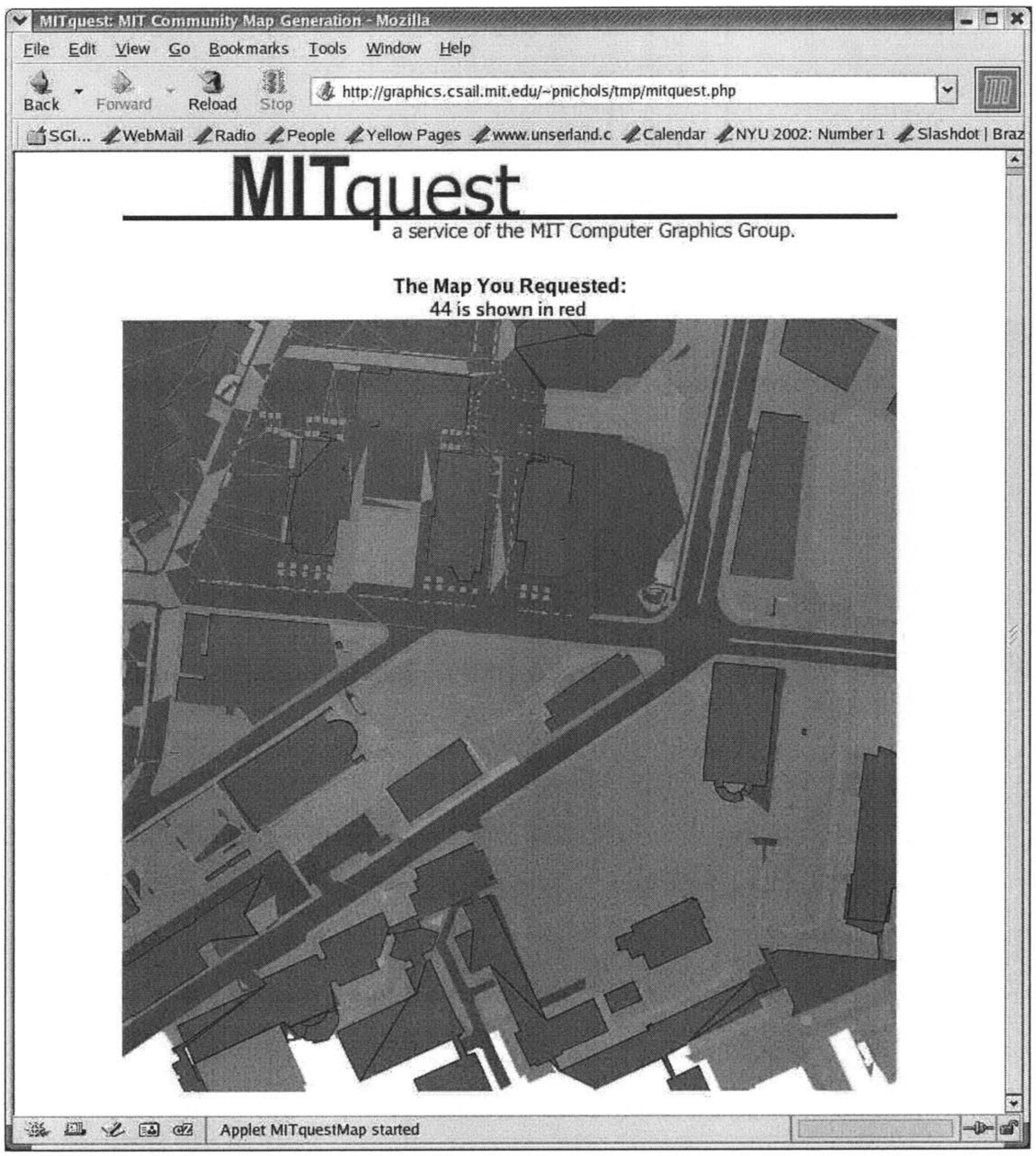

MITquest: MIT Community Map Generation - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back
Forward
Reload
Stop
http://graphics.csail.mit.edu/~pnichols/tmp/mitquest.php
SGI...
WebMail
Radio
People
Yellow Pages
www.unserland.c
Calendar
NYU 2002: Number 1
Slashdot | Braz
MITquest
a service of the MIT Computer Graphics Group.
The Map You Requested:
44 is shown in red
Applet MITquestMap started

Figure 4-6: MITquest Map Display. The requested building is shown in red, with streets, sidewalks, grass, and other buildings in its vicinity displayed.

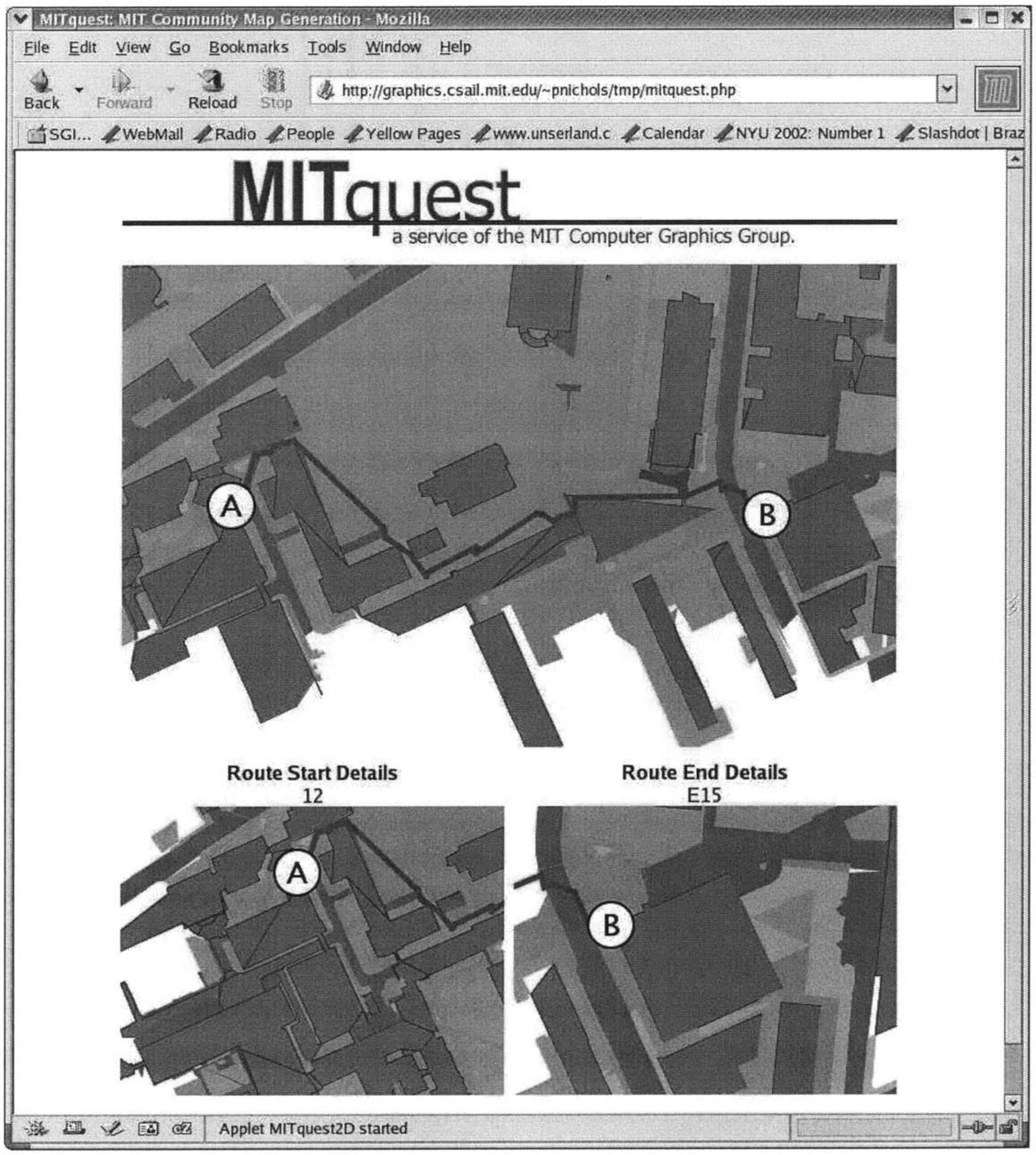

MITquest: MIT Community Map Generation - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://graphics.csail.mit.edu/~pnichols/tmp/mitquest.php
SGI... WebMail Radio People Yellow Pages www.unserland.c Calendar NYU 2002: Number 1 Slashdot | Braz
MITquest
a service of the MIT Computer Graphics Group.
A
B
Route Start Details
12
Route End Details
E15
A
B
Applet MITquest2D started

Figure 4-7: MITquest Route Display. The whole route is shown at top, with detailed views of the start and end of the route shown below.

Chapter 5

Active Signage Physical Design

This chapter addresses the physical design of location-aware active signs. We present the physical requirements for the hardware and infrastructure of active signs, discuss the tablet PC platform we chose, and explore alternative designs.

5.1 Design Requirements

The task of designing a physical form for an active sign requires the consideration of aesthetic and sociological factors, as well as more practical physical, budgetary, and technical constraints. [10] Our objective was to find a single universal physical design which could be constructed at minimal cost and placed in many location across MIT's campus. Correspondingly, we considered the following factors when selecting a hardware platform for active signs:

- Cost – An obvious objective was to minimize the cost of platform, with a target price of under $1000 per sign.

- Size – We wanted to select a platform with sufficient screen resolution to display detailed maps and routes, but was physically small enough to be easily mounted on walls.

- Hardware – All of the route-finding algorithms and methods discussed in this thesis require sufficient RAM and CPU capacity to load and display maps and routes.

- Connectivity – The client-server architecture we have chosen for route generation requires at least an Ethernet network interface, and preferably wireless Ethernet connectivity.
- Interactivity – Users need a way of interacting with the active sign, as show in Section 4. We considered many mode of interaction, including keyboards, mice, and touch screens.

Based on these requirements, we evaluated a number of physical form factors, including:

1. Wall-mounted laptops
2. Wall-mounted table PCs
3. Wall-mounted plasma/LCD displays, with an attached PC
4. Digital projectors with an attached PC

After considering these options, we selected the tablet PC form factor, purchasing two ViewSonic ViewPad 1000 tablets [12] as initial sign prototypes. The ViewPad combines a number of attractive features for use as an active sign, including wireless Ethernet, a 10.4" touch pad screen, on-board camera, microphone, and a small physical footprint. We found the ViewPad to be a reasonably-priced, fully functional hardware platform for location-aware active signage.

5.2 Physical Design

Active signs are physically embodied in wall-mounted tablets PCs. As seen in Figure 5-1, maps, routes and events are displayed on the 10.4 inch touchscreen display. Users can interact with signs in a number of ways, primarily by touching

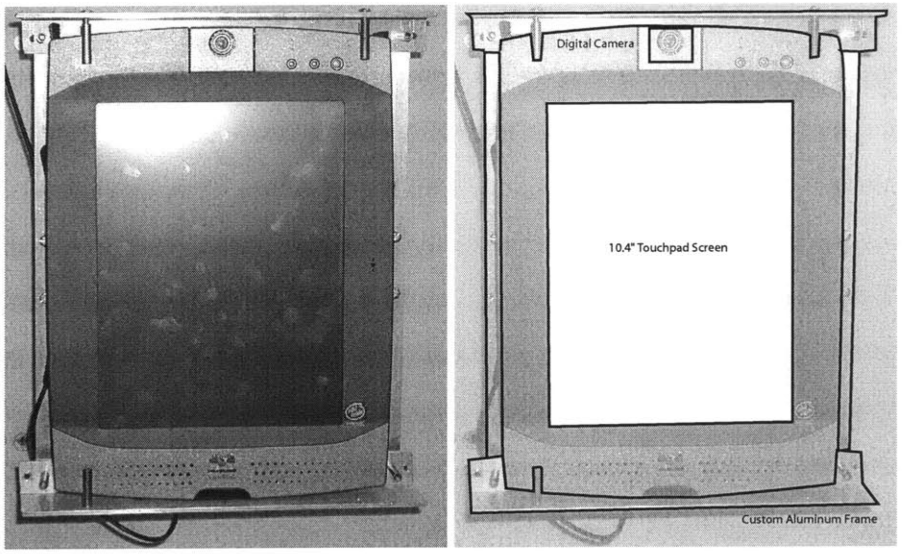


Figure 5-1: Front View of Active Sign. Note the custom aluminum frame, touch-screen display, and embedded digital camera.

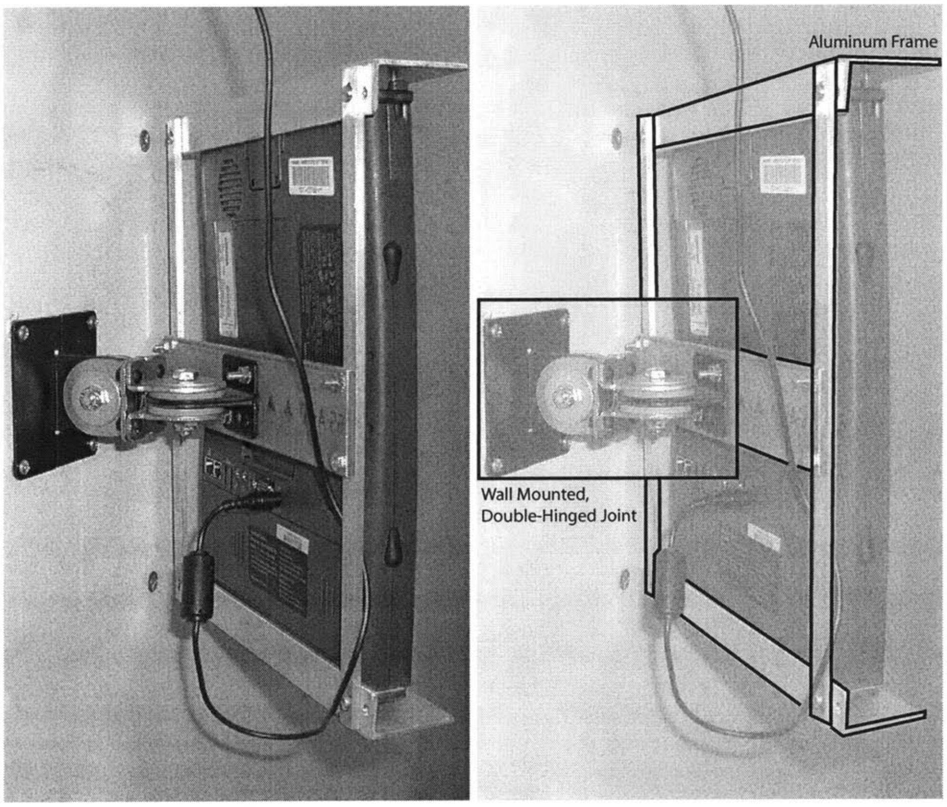


Figure 5-2: Profile View of Active Sign. Note the backside of the custom aluminum frame and the double-hinged joint mounted to the wall.

buttons on the screen, which does not require a special stylus or other input device. Each sign also has a set of speakers, microphone, and web cam as additional interface mechanisms.

Signs are physically mounted on the wall with a custom aluminum frame made in the MIT CSAIL machine shop. The frame serves a both a mounting harness to physically attach the sign to the wall and a theft-prevention device, with signs secured between a set of screws.[1] The frame is mounted to the wall using a double-jointed housing as shown in Figure 5-2. The armature lets users pivot the sign horizontally and vertically, comfortably accommodating those in wheelchairs while maintaining a minimal horizontal sign footprint.

5.3 Software Security

A final "physical" consideration for sign design was locking down access to the software, files, and operating system running on the sign. Unfortunately, the only operating supported on the ViewPad 1000 is Microsoft Windows XP, which has limited capabilities for totally securing the physical machine from a malicious user. We found that by creating a custom user with limited file and application permissions we could suitably restrict most users from tampering with the software running on the active signs. If a large number of signs were to be deployed in a widespread physical network, however, more effort would need to be invested in totally locking down illicit, unintended use of the signs.

[1]A determined thief could, of course, either remove the screws or cut through the aluminum frame with a saw. For more public installations of active signs (the models shown are in the MIT Computer Graphics Group lab), we would use more secure methods of affixing signs to the wall.

Chapter 6

Future Work and Conclusion

This chapter presents areas of future work for this thesis, including suggestions for enhanced route-finding, event aggregation, map usability, and thoughts for deploying a large network of distributed active signs. This chapter and this thesis are concluded with closing remarks.

6.1 Shortcomings

Location-aware active signs in their current form suffer from a number of shortcomings. Two major shortcomings – the problem of campus data generation and position acquisition – are discussed below.

One problem area is the acquisition of physical campus data, which is manually derived from a corpus of centrally-maintained MIT floor plans. Obtaining physically-accurate models of campus is a time-consuming, difficult task which has required the development of numerous custom tools, and which has yielded imperfect campus data. A further complication is that although we can continuously hone our automated tools, the real world is in constant flux. The MIT Department of Facilities updates a central corpus of floor plans, and these changes ultimately filter down through a pipeline of programs to update our virtual representation of the MIT campus. Unfortunately, our approach (at best) lags behind the real world by several weeks, as it takes time for these centrally-maintained

CAD files to be updated with new construction information. Thus one major shortcoming of our approach are the difficulties inherent in using a secondary source of data (CAD files) to generate models when they are not always up-to-date and often contain flaws.

A second issue is that there is currently no way for a client application to automatically obtain its position from the environment; at present this must be specified by user either per route query (as with MITquest), or at start-time (as with a location-aware active sign). In the future, position systems like Cricket [7] will provide both position and orientation, but such capabilities have been deployed only in limited portions of campus.

6.2 Future Work

There are a number of fronts on which this thesis can be extended. Here, we consider a few areas of such future work, including enhanced route-finding, improved map usability, and more extensive event aggregation.

6.2.1 Enhanced Route-finding

There are a number of optimizations which could increase the speed and fidelity of route-finding in the location server.

The fundamental algorithm used to find routes is Dijkstra's algorithm (discussed in Section 2.5), which tends in practice to have poor performance when searching for routes between distant spaces. New MIT students use a very simple algorithm when divining routes between classrooms – if their destination is in a different building than their current location, they typically exit the building as quickly as possible and then locate the building of their destination.

The methods presented in Section 2.5 rely on brute-force searching of a campus graph to arrive at the same result. One heuristic we could add to route generation in the location server is to break paths into distinct components, and append the results of searches on these smaller sub-queries. For example, consider a route which goes between two buildings. We could break such a route into a number of smaller routes:

- From the start space to the building exit
- From the start space's building exit to the destination space's building entrance
- From the destination space's building entrance to the destination building

A number of heuristics could speed up each of these smaller searches, enabling better route generation overall.

6.2.2 Map Usability

We have touched on the differences between merely valid routes and usable maps in Sections 2.5.2 and 4.1, and there is much more work to be done on this front. Research [1] shows that maps and routes with high fidelity are often less usable and useful than stylistic representations of the same data. We have presented methods for extracting essential information from routes, but many extensions can be made to the simple techniques explored in this thesis.

In particular, selectively clipping walls in 3D route maps could greatly enhance usability. Magazines and newspapers often use a standard 3/4 cutaway view of buildings to display both interior and exterior structure – an approach which could greatly enhance the usability of our 3D route-finding techniques.

6.2.3 Event Aggregation

As discussed in Section 3.5.3, we currently infer events from two sources: the MIT events calendar, and the LCS events page. Although many public events are published on convenient, well-formatted sources like the aforementioned two, many campus events are distributed through another electronic medium: email.

One viable extension to the event server is to set up an email account, subscribe the account to a number of popular campus event mailing lists, and to thus infer events from emailed event descriptions sent to the account. While addressing the challenges of inferring basic event data from emailed descriptions poses many challenges, adding this feature would greatly expand the set of events available to the EventServer, making active signs useful community tools.

6.3 Conclusion

We have presented an architecture and methods for generating two- and three-dimensional route maps, a collection of general Java classes for constructing such maps, and provided exemplary applications that provide useful services to the MIT community.

We have implemented route-finding in a client server architecture, building a Java location server with substantial performance optimizations, including route memoization, space convexity checking, and portal path pre-computation inside spaces. By using a rich route model, we further enable constrained route generation in Dijkstra's algorithm, letting client applications specify constraints to accommodate a variety of needs.

Map display is implemented in two Java classes that make map and route display abstract for developers and accessible for end-users. We have built client applications using our base classes, including location-aware active signs and MITquest, which provide maps and other services. Active signs have been further endowed with knowledge of events around MIT's campus, enabled through a central event server and methods for inferring events from MIT community web pages.

We hope that the methods, applications, and improvements presented in this thesis are of use to members of the MIT community and motivate the further development of location-aware applications. This thesis stands as a useful proof-of-concept for a set of location-aware, active signs.

Appendix A

Project Build Instructions

This chapter describes how to checkout, build, and execute the applications we discuss in this thesis, including the LocationServer, EventServer, TextSign, GraphicsSign, and MITquest applications. These instructions assume an account with the MIT Computer Graphics Group, and membership in the *graphics* and *city* groups. These instructions also assume a Linux command-line environment, running under the tcsh shell.

A.1 Checkout and Build Instructions

First, check out the CVS source tree of the `walkthru/mit` project. Do this with:

```
% setenv CVSROOT /d9/projects/
```

Make sure that this environment variable is set by checking the environment, do this with:

```
% env | grep CVSROOT
```

and verify that the CVSROOT environment variable is properly set. Next, move to the directory where the `walkthru` source tree will be hosted. To checkout the entire directory, type:

```
% cvs checkout -P walkthru/mit
```

A.2 Invoking Applications

A.2.1 LocationServer

The location server is invoked with the following command-line arguments:

```
% LocationServer mode file building1 ...  buildingN
```

The `mode` parameter specifies which mode the location server is run in; there are either “batch processing mode” with the parameter `BATCH_OUT` or “quick start” mode, with the parameter `QUICK_START`. In batch processing mode, the location server load and pre-processes the list of named buildings, and then outputs its pre-processed state to the specified `file` before binding to Java RMI port. The `building` parameter is simply the path to the root building name for one or more building data files. For example, `/scratch/pnichols/bmg/mit_13` is the root name for `mit_13.spaces` and `mit_13.portals`, the two files containing all room geometry for building 13. The actual format of these space and portal files is the subject of Appendix C.

To actually compile and invoke the location server, first move to the root directory where the `walkthru` source tree has been installed. Then:

```
% cd walkthru/mit/src/LocationAware/locationserver
% rmiregistry &
% make ; make full
```

The location server will then load campus data from the local file system, process campus floor plans, and then publish itself on port 5432 (the default Java RMI port). The set of buildings to load, and their locations, is in the Makefile itself. Making the location server network-accessible may require that the server hosting it explicitly open port 1099 to the outside world depending on firewall and other security settings. Compilation requires that the Java programs `javac` and `rmic` are both installed and accessible. To check whether these applications are in the path, type:

```
% which javac
```

If javac is not found, try:

```
% locate javac | grep bin
```

This should locate the appropriate directory to add to the PATH environment variable.

A.2.2 EventServer

To invoke the event server, first move to the root directory where the walkthru source tree has been installed. Then:

```
% cd walkthru/mit/src/LocationAware/
% setenv CLASSPATH {$PWD}
% javac eventserver/*.java
% java eventserver.EventServer INTERACTIVE
```

The event server will not compile or run without the Java CLASSPATH environment variable set to walkthru/mit/src/LocationAware.

These instructions run EventServer in "interactive" mode, which displays the GUI referred to throughout this thesis. To run the EventServer as a background process, replace INTERACTIVE with SERVER when invoking the application from the command line. Once the EventServer application is running, press the "Probe Events Database" button to automatically acquire a set of events from LCS events web page and the MIT community events calendar. Event databases can also be saved to disk and loaded using the corresponding GUI buttons.

A.2.3 TextSign

Invoking a TextSign requires an available, accessible event server and location server. The TextSign application must also have the physical location of the sign entered as an initial command-line parameter. This physical location is specified as a space name, using the naming convention adopted in this thesis, discussion in Section 2.4.

In the future, this will alternately be provided by a system like Cricket, and will likely be specified as a pose, including both position and orientation. Should this change, the `getPhysicalPosition()` method should be modified appropriately.

TextSign is called with:

```
% TextSign location eventserver locationserver
```

To run the TextSign application (for example) with an event server running locally and a location server running on graphics.csail.mit.edu:

```
% cd walkthru/mit/src/LocationAware/
% setenv CLASSPATH {$PWD}
% javac textsign/*.java
% java textsign.TextSign mit_38#1#00LA#LOBBY localhost
  graphics.csail.mit.edu
```

A.2.4 GraphicsSign

The GraphicsSign component requires that Java3D be locally installed and in the `PATH` environment variable. GraphicsSign takes three command-line arguments – the location of a master TextSign, and the LocationServer to provide geometry information.To run GraphicsSign, type:

```
% cd walkthru/mit/src/LocationAware/
% setenv CLASSPATH {$PWD}
% javac graphicssign/*.java
% java graphicssign.GraphicsSign localhost graphics.csail.mit.edu
```

In this example, the application would run with a location server running on `graphics.csail.mit.edu` and a textsign running locally.

A.2.5 MITquest

Installing and running the MITquest applet requires a web server such as Apache with PHP4 installed. To obtain and compile MITquest, type:

```
% cd walkthru/mit/src/LocationAware/
% setenv CLASSPATH {$PWD}
% cd MITquest/
% javac *.java
```

To access MITquest from the web, create a symbolic link to the MITquest directory from a web accessible directory. For example, on the graphics server one would:

```
% cd /citypub/www/city/bmg/
% ln -s  /walkthru/mit/src/LocationAware/MITquest mitquest
```

This would make MITquest available on the web at
http://city.csail.mit.edu/bmg/mitquest

Appendix B

API Documentation

This chapter briefly discusses the `LocationServer` and `DisplayLayer` interfaces, providing high-level insight into their use and purpose. It then provides the outline for a reference client which makes use of these components to provide route-finding services to an end user.

B.1 The LocationServer Interface

The `LocationServer` interface, shown in Figure B-1, exposes high-level geometry and route-finding capabilities to client applications.

This interface is implemented by the `JavaLocationServer` application, which is a Java RMI server. Any clients making use of a location server must instantiate this class locally by using the Java lookup method to locate a network-accessible

```
public interface LocationServer {
  public Space getSpace(Position position);
  public Space getSpace(String name);
  public Portal getPortal(Position position);
  public Portal getPortal(String name);
  public Route getRoute(Route route);
}
```

Figure B-1: LocationServer Interface.

LocationServer bound to the machine at `server` (such as `graphics.csail.mit.edu`).

```
try {
  String name = "//" + server + "/LocationServer";
  this.locationServer = (LocationServer) Naming.lookup(name);
} catch (Exception e) {
  e.printStackTrace();
}
```

Once a client has instantiated a local copy of the LocationServer class, the client then uses the API in Figure B-1 to build maps and display routes. Typically, a client must first build an internal map by specifying an initial space (using the `getSpace()` method), then recursively query for spaces adjacent to the initial space. Each space maintains a list of the names of its adjacent spaces, so a client may obtain the actual adjacent spaces by recursively querying the LocationServer with the `getSpace()` method. Clients can restrict maps by selectively adding new spaces, such as by only adding spaces on a certain floor of a certain building.

Once a client has built a map by collecting a set of spaces and portals, it can request routes from the `LocationServer`. This is done with the `Route` class, which specifies a set of constraints on a route, as well as various collections of points, spaces, and portals which satisfy the route constraints. The `Route` constructor takes:

```
public Route(Space source, Space dest, String type);
```

Where source is the starting Space, dest in the destination Space, and type specifies Route constraints, represented as a static String in the Route class. To generate a valid route, clients must first create a local `Route` object, and then call the `getRoute()` method, saving the returned route. For example, a client might generate a route like this:

```
Space s1 = this.locationServer.getSpace(‘‘mit_NE43@2@255@OFF’’);
Space s2 = this.locationServer.getSpace(''mit_NE43@5@501@OFF'');
Route myRoute = new Route(s1, s2, Route.ROLLING_ROUTE);
myRoute = this.locationServer.getRoute(myRoute);
```

The returned object contains the collection of points, portals, and spaces which optimally satisfy the route constraints. If no valid route can be found by the `LocationServer`, the returned object is null.

B.2 The DisplayLayer Interface

The `DisplayLayer` interface, shown in Figure B-2, is used to instantiate and control very general 2D and 3D map and route display objects, which themselves extend the Javax.Swing.JPanel class. Implementors of the `DisplayLayer` class promise basic map and route display functionality, and can be used interchangeably in applications like MITquest and active signage.

```
public interface DisplayLayer {
  public DisplayLayer(LocationServer ls, Space initial);
  public boolean drawRoute(Route r);
  public boolean markSpace(Space s, String text);
  public void clearAll();
}
```

Figure B-2: DisplayLayer Interface.

`DisplayLayer` classes are instantiated with references to an initial space (which serves as the initial space used to build an initial map, as detailed in Section B.1) and a `LocationServer` to provide geometry information. Clients are must also provide methods for displaying routes, marking spaces with optional text, and for clearing all displayed routes and markings.

Appendix C

LocationServer File Format

This chapter documents the file format used as input by the LocationServer application, which is discussed at length in Section 3.3. The location server takes a set of file names, passed as command-line arguments, as arguments to load and process rooms and portals. The location server is passed the full path to a root filename for each room to be loaded; a sample set of this data is located on the graphics file system at `/scratch/pnichols/bmg/`. Section A.2.1 contains more information on invoking the `LocationServer` application.

Each building has two associated files – a file containing a room geometry (which has the extension `.spaces`) and a file containing room connectivity information (which has the extension `.portals`). For example, MIT building 13 would have all room data stored in `mit_13.spaces` and all room connectivity data in `mit_13.portals`.

C.1 Room File Format

Building room geometry is stored in a simple, plain text format. Each space is represented as a single line in the file, in the following format:

```
SPACE_NAME CONTOUR_POINTS | SPACE_TRIANGULATION
```

The `SPACE_NAME` follows the format:

```
BUILDING_NAME#FLOOR_NUMBER#ROOM_NUMBER
```

The `CONTOUR_POINTS` for a space represent the the 2d footprint of the space, represented as an ordered list of 3d points. The list should close, e.g. the last point of the ordered points should be the same as the first point. For a space with n points, the format is:

$$\{p_{x_1} p_{y_1} p_{z_1}\} \ldots \{p_{x_n} p_{y_n} p_{z_n}\}$$

The `SPACE_TRIANGULATION` for a space is the CDT triangulation of the space, represented as sets of points enclosed by braces. For a space with m triangles, the format is:

$$\{\{t_{1_{x1}} t_{1_{y1}} t_{1_{z1}}\} \ldots \{t_{1_{x3}} t_{1_{y3}} t_{1_{z3}}\}\} \ldots \{\{t_{m_{x1}} t_{m_{y1}} t_{m_{z1}}\} \ldots \{t_{m_{x3}} t_{m_{y3}} t_{m_{z3}}\}\}$$

C.2 Portal File Format

Space adjacency information is also stored in a plain text file. Each portal is represented as a single line in the file, in the following format:

```
PORTAL_NAME PORTAL_TYPE SPACE_1 SPACE_2 PORTAL_SHAPE
```

The `PORTAL_NAME` is simply a globally unique string identifying the portal, which in practice is usually a non-negative integer.

The `PORTAL_TYPE` represents what type of space portal connects the two named Spaces. Legal portal types include `STAIR_UP`, `STAIR_DOWN`, `ELEV_UP`, `ELEV_DOWN`, `OUTSIDE`, `WINDOW`. All portal types are public, final member variables of the Portal class.

Each portal connects two spaces, whose ids are listed as `SPACE_1` and `SPACE_2`. These adjacencies are one-way only, which means that most (but not all) portals have a twin which connects the two spaces in the other direction. This reflects the fact that some doors (such as emergency exit doors) are one-way only.

Each portal also has a physical footprint, as represented by the `PORTAL_SHAPE`, which is a set of four points representing the quadrilateral outline of the portal. This takes the form:

$\{p_{1_x} p_{1_y} p_{1_z}\} \ldots \{p_{4_x} p_{4_y} p_{4_z}\}$

Bibliography

[1] Agrawala, Maneesh. *Visualizing Route Maps.* PhD Thesis, Stanford University, Palo Alto, CA, December 2001.

[2] Bell, Jason Murray. *An API for Location-Aware Computing.* M.Eng Thesis, Massachusetts Institute of Technology, Cambridge MA, January 2003.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms,* second edition. The MIT Press and McGraw-Hill, 2001.

[4] Kulikov, Vitaly. *MIT Campus Basemap Coloring.* M.Eng Thesis, Massachusetts Institute of Technology, Cambridge, MA. In progress.

[5] Lischinski, Dani. *Graphics Gems IV, Chaper I.5: Incremental Delaunay Triangulation.* Academic Press, 1994.

[6] Lewis, Rick. *Generating Three-Dimensional Building Models from Two-Dimensional Architectural Plans.* Master's Thesis, UC Berkeley, Berkeley CA, 1996.

[7] Nissanka B. Priyantha, Anit Chakraborty, Hari Balakrishnan. *The Cricket Location-Support system,* Proc. 6th ACM MOBICOM, Boston, MA, August 2000.

[8] Seth Teller, Jiawen Chen, Hari Balakrishnan. *Pervasive Pose-Aware Applications and Infrastructure,* IEEE CG&A, July/August 2003.

[9] Seth Teller, Tom Funkhouser, Delnaz Khorramabadi, Carlo Sequin. *The UCB System for Interactive Visualization of Large Architectural Models.* Prescence, Vol. 5, No. 1, Winter (January) 1996.

[10] Van Kleek, Max. *Intelligent Environments for Informal Public Spaces: the Ki/o Kiosk Platform.* M.Eng Thesis, Massachusetts Institute of Technology, Cambridge, MA, February 2003.

[11] Xu, Keyuan. *Three-Dimensional Reconstruction of Campus Buildings.* UROP Report, Massachusetts Institute of Technology, Cambridge, MA, 2001.

[12] ViewSonic ViewPad 1000 Factsheet.
http://www.viewsonic.com/products/tablet_pc_viewpad1000.htm

6075 - 21