

Federating Data Sets for Use In a Map Context

by

Grayson P. Giovine

B.S. Computer Science

B.S. Mathematics

Massachusetts Institute of Technology, 2005

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

JUNE 2006

© 2006 Grayson P. Giovine. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and
distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.

Signature of Author: _____
Department of Electrical Engineering and Computer Science
May 26, 2006

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

Accepted by: _____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Federating Data Sets for Use In a Map Context

by

Grayson P. Giovine

Submitted to the Department of Electrical Engineering and Computer Science on May 26, 2006,
in partial fulfillment of the requirements for the degree of
Master of Engineering in Computer Science And Engineering

Abstract

The goal of this project is to federate data sets containing location information such that they can be used in a map context. There are numerous examples of these data sets around the MIT campus: asset data with the Property Office, equipment data with Facilities, ShuttleTrack data, student/staff contact information, and hazardous materials data. In addition, there are many objects around the MIT campus that are not kept in a database or formally maintained in any way, and we would like to capture these in our system as well.

In order to do this, we must first create tools and methods for capturing these location-based data sets, enabling us to bring the data into our system in a structured manner. Once we have this data “federated”, we can use the interactive map infrastructure we have developed to create visualizations. These visualizations can range in complexity, from simple one data set visualizations, to visualizations of multiple related data sets.

Furthermore, we would like to generalize these visualizations such that little or no manual tweaking is necessary for any federated data set. This would allow a data administrator to easily create a customized visualization for his/her federated data set.

We will provide the functionality for federating and visualizing data sets in an API. This will help streamline and automate the integration process, and will allow this system to be easily expanded.

Thesis Supervisor: Seth Teller

Title: Associate Professor of Computer Science and Engineering

Acknowledgments

I would like to thank my thesis advisor, Seth Teller, for his creativity and involvement in this process. He was always able to see beyond the immediate scope of the code I was getting my hands dirty with, and his vision was an inspiration.

I would also like to thank my teammates on the BMG team – Yoni Battat and Emily Whiting. Without their tireless efforts, the infrastructure on which my thesis is based would not exist.

Thanks also goes to Professor Lawrence Susskind, who I TAed for in Spring 2006. By providing me the opportunity to TA his course on negotiation, I was able to easily continue work on this thesis.

Milly Lang, a friend of mine, also deserves my appreciation. She read through a number of thesis drafts and provided helpful comments.

Finally, I would like to thank my parents, who have always done their best to provide me opportunities to succeed.

Table of Contents

1. Introduction

- 1.1. Motivation
- 1.2. Contributions
- 1.3. MIT WikiMap Overview
- 1.4. Thesis Overview

2. Background

- 2.1. Existing Infrastructure
 - 2.1.1. Data Structures
 - 2.1.2. Location Formats
 - 2.1.3. Google Maps
 - 2.1.4. MIT Maps
 - 2.1.5. IMS
- 2.2. Building Model Generation
 - 2.2.1. Data Model
- 2.3. Data Sets
 - 2.3.1. Assets
 - 2.3.2. Equipment
 - 2.3.3. Schedules
 - 2.3.4. People
 - 2.3.5. Hazardous Materials
 - 2.3.6. Colloquial Information
 - 2.3.7. Other Data Sets
- 2.4. Property Office
 - 2.4.1. Asset-tracking

3. Data Model

- 3.1. Existing Pipeline
- 3.2. New Pipeline
- 3.3. Spatial Hierarchy
- 3.4. Spaces
 - 3.4.1. XML Representation
- 3.5. Portals
 - 3.5.1. XML Representation
 - 3.5.2. Dangling Portal Connector
- 3.6. Federation From IMS
 - 3.6.1. Contour Drawing
 - 3.6.2. Image Caching

3.6.3. Point Queries

4. Approach

4.1. Overall Approach

4.2. Data Representation

4.3. Data Federation

4.3.1. Assets

4.3.2. Equipment

4.3.3. Schedules

4.3.4. People

4.3.5. Hazardous Materials

4.3.6. Colloquial Information

4.3.7. Other Data Sets

4.4. MIT WikiMap

4.5. Property Tool

4.6. Other Applications

4.7. Back-End Design

4.7.1. Table Descriptions

4.7.2. History

4.8. Searching For Objects

4.8.1. Constrained to One Data Set

4.8.2. Across Multiple Data Sets

4.8.3. Drop-down

5. MIT WikiMap

5.1. Problem

5.2. Overall Design

5.2.1. Front-End / Back-End Interaction

5.2.2. Implementation

5.3. User Interface Design

5.3.1. Map Browsing

5.3.2. Adding New Objects

5.3.3. Editing Objects

5.3.4. Searching For Objects

5.3.5. WikiMap Categories

5.3.6. Storing User Information

5.3.7. System State

5.3.8. Deterring Vandalism

5.4. User Testing

5.4.1. Procedure

5.4.2. Results

5.5. Reflection

6. **Property Tool**

6.1. Problem

6.2. Overall Design

6.3. Administrative User Interface Design

6.3.1. Connecting to SumProp

6.3.2. Visualizations

6.3.3. Emailing Users

6.3.4. Grouping Assets

6.3.5. Confirming Updates

6.3.6. Following-up With Users

6.4. End-User Interface Design

6.4.1. Visualizations

6.4.2. Responding To Inquiries

6.5. User Testing

6.5.1. Procedure

6.5.2. Results

7. **Other Applications / Visualizations**

7.1. Visualization Builder

7.1.1. XML Representation

7.1.2. Visualization Types

7.2. Category Icon Tool

8. **Conclusions & Future Work**

8.1. Shortcomings

8.2. Future Work

8.2.1. MIT WikiMap Improvements

8.2.2. Application Builders

8.2.3. Advanced Visualizations

8.3. Conclusions

Bibliography

A. Project Build Instructions

B. API Documentation

C. Permalinks

D. Contact Information for Data Sets

E. Algorithms

E.1 Point-in-Polygon Test

Chapter 1

Introduction

This chapter presents the motivations for this project and the contributions it seeks to make. We also briefly provide an overview of the MIT WikiMap, an application that nicely demonstrates the framework we have developed. Finally, we present an overview of the thesis as a whole.

1.1 Motivation

The motivation for this project stems from the desire to bring together a number of distinct location-based data sets from around the MIT campus in a structured way. As map applications have become more interactive and complex, it has become easier to create helpful visualizations of data. If we are able to devise a structure to “federate” this data, we can potentially create powerful applications that can visualize information from multiple data sets and perform analysis that would not be possible in the data sets’ current isolated form.

Many of the data sets on the MIT Campus actually have been brought into a common infrastructure – the MIT Data Warehouse. This has been an effort to standardize the way data at MIT is stored and accessed. This effort has made it easier to do analysis using multiple data sets. To give a simple example, to find all the hazardous chemicals residing in the “Uncas A Whitaker Building”, we could do a join on the building number between two tables. This would result in “Uncas A Whitaker Building” (from the Buildings table) being joined to rows in the hazard table where the building number is 56, and thus our query could be successfully answered.

However, not all data sets reside in this infrastructure – examples include asset data with the Property Office, equipment data with Facilities, schedules data with the Registrar, ShuttleTrack data, MIT events feeds, and others. This is a problem because it greatly restricts the variety and complexity of analysis we are able to perform.

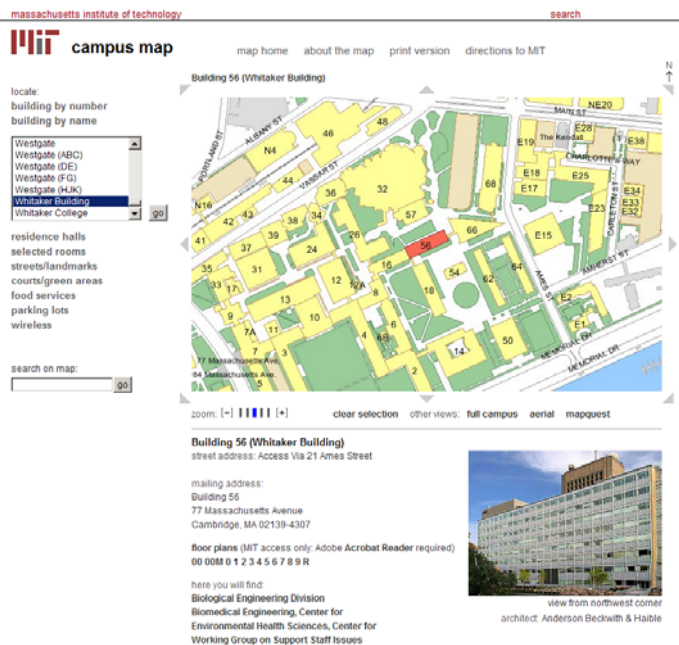
One of the main goals of this project, consequently, is to make it easy to combine all location-based data sets, no matter where they reside. We would also like to develop a framework for this “data federation”. With a minimal number of restrictions, we would like to take any data set and be able to produce interesting and useful visualizations and applications.



There have been efforts in the past to create visualizations of location-based data sets in a map environment. Figure 1-1 shows a map of the MIT Campus' wireless coverage.

Figure 1-1: Wireless coverage on the MIT Campus.

Though this visualization is certainly helpful, it is not interactive or fine-grained enough to be especially useful. Figure 1-1 is actually demonstrating a JPEG image, not an interactive map.



There have also been modest efforts to link location-relevant data on the MIT Campus. WhereIs, a map service for MIT, is one instance where this occurred. Figure 1-2 demonstrates a search for the “Whitaker Building” – consequently, a highlight of building 56 appears. A link between “Whitaker Building” and “56” is needed in order for this to happen, and such a link is contained in a row of the “Buildings” table in the MIT Data Warehouse.

Figure 1-2: Searching for the Whitaker Building on WhereIs.

Outside of the MIT Campus, there have been more ambitious efforts to visualize location-based data sets in interactive maps. Since Google Maps had its beta launch in February 2005, countless people have written “mash-ups” of the map, reverse-engineering the Google code in order to build layers on top it. Examples include the Chicago crime report web page [28] and a mash-up that places Craigslist’s real estate listings on a map of the US, as demonstrated in Figure 1-3 [16].

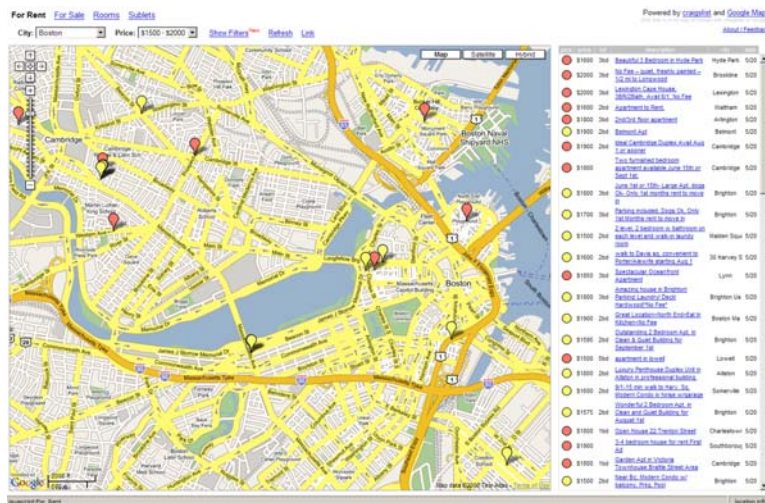


Figure 1-3: Apartments for rent in Boston/Cambridge - <http://www.housingmaps.com>.

Now, it is even easier to build such applications because Google published the API for its map tool, allowing people to create visualizations without reverse engineering. This became a motivator for creating an interactive map tailored to the MIT Campus – MIT Maps. The application was designed with modularity in mind, and an API was created, which made it easy to extend in the future. It also facilitated layering objects on the map.

Given the ability to visualize objects on the campus in an interactive way, the importance of federating data came to the fore. Though we could certainly create a visualization of a data set using this framework, it would take a substantial amount of work to do so in a robust manner, and would not necessarily be easily extended to other data sets. Therefore, in order to visualize the wealth of location-based data on the MIT Campus, we needed to think carefully about the similarities between location-based data sets, and create a framework for federating them.

In addition, it is important that we specify a clean and easy-to-use API that can be picked up by any data owner and used in order to add data to this system. The API will also have methods for easily visualizing the data that has been federated.

When we speak about “visualizations”, we are referring to two principal types: discrete and continuous. Discrete visualizations involve placing objects at locations on the campus – similar to what was done in Figure 1-3. A continuous visualization is more complex, such as the

coloring of a map. For example, the police department may want to view a colorized representation of the dollar amount stolen from each area on the MIT campus. This coloring would ideally be smooth, such that there is a gradient of color intensity between where a crime occurred and more distant locations. Figure 1-4 demonstrates an example of this, though for the city of Chicago rather than MIT.

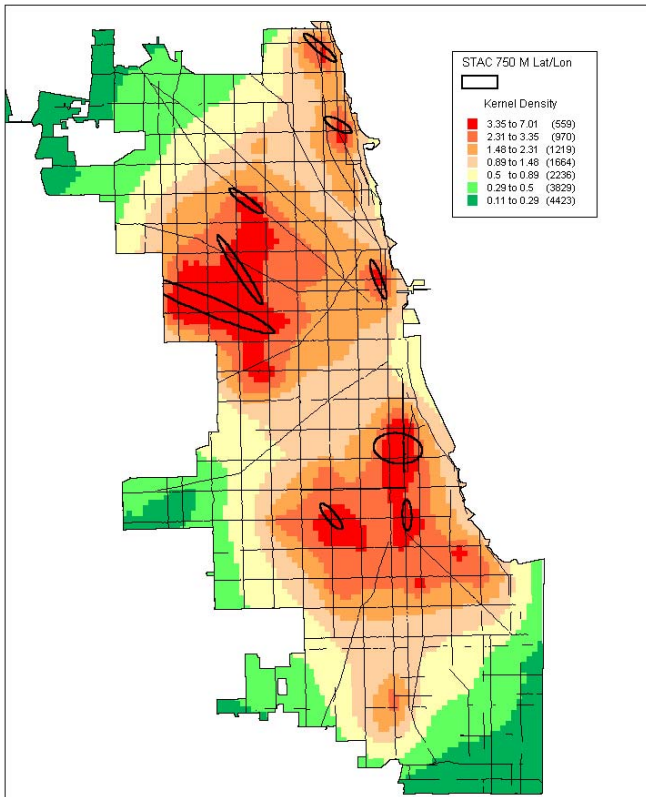


Figure 1-4: Crime density map of Chicago.

1.2 Contributions

This thesis makes the following contributions:

- A framework for federating location-based data sets to bring them into a map context
- A framework for hooking into each of these data sets easily via searching
- A framework for representing objects in this system that is flexible enough to allow any location-based data to be represented
- MIT WikiMap, a website for sharing and collaboratively editing points of interest around the MIT campus
- Property Tool, an application for helping the Property Office do their asset inventory
- Visualizations of other data sets that have been federated, including equipment, people, schedules, and hazardous materials data.

1.3 MIT WikiMap Overview

The MIT WikiMap is a tool for sharing and collaboratively editing points of interest around the MIT campus. It provides a way to learn about a piece of artwork, a bike rack, or a tidbit of trivia specific to a location on campus. Users may select assorted object categories to view on the map, and once these objects appear, can click on them to view more information. A user may, for example, want to look up all the bike racks near 10-250 in order to park his/her bike before class. By selecting the Bike Racks category, the bike racks in the desired area can be identified.



Figure 1-5: MIT WikiMap interface.

The MIT WikiMap was built by calling functions provided by the MIT Map. The abstraction layer between the two makes it such that development can be (and was) done in parallel. Furthermore, because of the level of abstraction already in place, this system can easily be extended to other data sets.

1.4 Thesis Overview

The second chapter presents background necessary to understanding the subject matter of this thesis. The third chapter discusses the data model the Building Model Generation group has developed, and the improvements that have been made to the existing infrastructure in place.

The fourth chapter discusses the approach used to federate multiple data sets for use in a map context, and the more specific approach used for each individual data set and application. The fifth chapter goes into detail on the MIT WikiMap, an application that applies the approaches discussed in the fourth chapter. The sixth chapter discusses the Property Tool, which is similar in nature to the MIT WikiMap, though geared towards a different audience and thus distinct in functionality. The seventh chapter presents other applications and visualizations created using this framework. The eighth chapter offers conclusions drawn as a result of the project, and where future work may lead us.

Chapter 2

Background

This chapter discusses the background needed to understand this thesis. First, we must understand the existing infrastructure prior to this project, both in terms of the data structures that are currently in place, as well as the mapping tools available on which to build. We will then more specifically discuss the location-based data sets on the MIT Campus. Finally, we will provide some background on the asset-tracking done by the Property Office, as this is essential to understanding the functionality of the tool that was built for them.

2.1 Existing Infrastructure

This project builds upon infrastructure developed in two realms. The first realm we will discuss is that of data structures, which pertain to both location-based data (the structures we are federating) and locations themselves (i.e. how are locations represented). The second realm is the infrastructure in place that allows for varying levels of interactivity within mapping tools. Thus, we will briefly discuss Google Maps, MIT Maps, and IMS, structures on which our interactive map interface is based.

2.1.1 Data Structures

As the global wealth of data has increased, the need for structure has become increasingly important. MIT has a large amount of data in its own right, and there have been efforts to better organize it. The MIT Data Warehouse has been at the forefront of this effort. It “provides the MIT community with integrated data from various administrative systems stored in one location” [23]. The Data Warehouse uses an Oracle database to store its data, and has standardized methods for access control and storage of data. This is convenient because it there is then a standard way to access a number of data sets.

As mentioned in the introduction, despite this effort, there are a number of data sets not in the Data Warehouse. These data sets are structured, though they are within different frameworks. Since they do not conform to the Data Warehouse standard, we need to be flexible in our approach to data federation. Further background on each of the individual data sets will be discussed in section 2.3.

2.1.2 Location Formats

Locations are represented in a variety of ways, and we must be able to interpret a wide range for this infrastructure to be most useful. There are four categories of location structures we would like to support:

- 1) Massachusetts state plane coordinates
- 2) Latitude-longitude (GPS) coordinates
- 3) MIT buildings, rooms, landmarks, and colloquial names
- 4) Street addresses

We will now briefly describe each of the four categories.

Massachusetts state plane coordinates:

“In the 1930s, the U.S. Coast and Geodetic Survey established a plane coordinate system for each of the 48 states. One to five zones were established in each state with a Lambert Conformal or a Traverse Mercator projection. The specific projection and the size of the zone was selected to fit the geometry of the state and to keep distortions at or below one part in 10,000” [27].

Since state plane coordinates require low distortion, state plane maps are no more than 158 miles across, and often have more than one projection to cover the entire area of the state. This is typically broken across county boundaries. In Massachusetts, state plane coordinates are based on two Lambert Conic projections, one for the *Mainland Zone* (most of the state) and the other for the *Island Zone* (Dukes and Nantucket Counties – the Elizabeth Islands and Martha's Vineyard, and Nantucket Island) [9]. Figure 2-1 demonstrates this division.

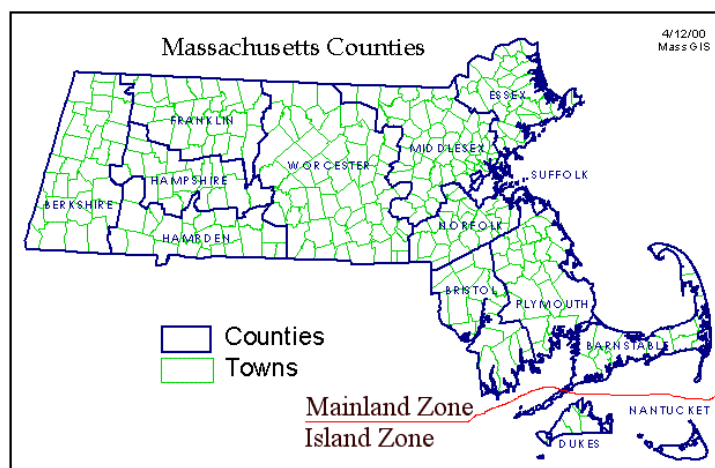


Figure 2-1: Massachusetts state plane projections – Mainland Zone and Island Zone.

IMS (see section 2.1.5) uses state plane coordinates. The extent of the MIT campus is from (693000,489000) in the lower left to (723000,504000) in the upper right. Since IMS uses this format, and it is highly accurate, we chose to use it as the default in our infrastructure. All formats are converted to Massachusetts state plane coordinates before being placed in our map applications.

Latitude-longitude (GPS) coordinates:

Latitude-longitude coordinates are a well-known format. Latitude is the angle between any point and the equator. Longitude is the angle east or west of the Royal Observatory in Greenwich, UK (this is at 0 degrees). Latitude and longitude can be divided into degrees, minutes, and seconds. There are numerous formats for this: 1) DM (Degree:Minute), 2) DMS (Degree:Minute:Second), and 3) DD (Decimal Degree). The most popular is decimal degrees, which is simply:

Decimal degrees = whole number of degrees + minutes / 60 + seconds / 3600 [10].

GPS receivers typically use latitude-longitude coordinates, so we will sometimes refer to this type as GPS coordinates.

Latitude-longitude coordinates are good for encoding a large area of spaces in the same system. Google Maps uses latitude-longitude coordinates in decimal notation.

MIT buildings, rooms, landmarks, and colloquial names

Examples of this include “Building 10”, “10-250”, “Killian Court”, “77 Massachusetts Avenue”, and “Whitaker Building”. In our data model, we store the Massachusetts state plane coordinates for each formal space name. We can also find the formal space name for a colloquial name in order to obtain its coordinates.

Street addresses:

These are commonly used in everyday life, but some conversion to a coordinate system must be performed in order to place street addresses on a map. We use a geocoder for this. Google uses a data vendor called Navteq for its geocoding, but there are a number of free services available on the web. Figure 2-2 demonstrates one such site – <http://geocoder.us/>.

Address	77 Massachusetts Ave Cambridge MA 02139
Latitude	42.359368 ° N 42 ° 21' 33.7"
Longitude	-71.094208 ° W 71 ° 5' 39.1"

Search for another address:

<input type="text" value="77 Massachusetts Avenue, Cambridge, MA"/>	<input type="button" value="Submit"/>
---	---------------------------------------

Figure 2-2: 77 Massachusetts Avenue, Cambridge, MA latitude-longitude coordinates – <http://geocoder.us/demo.cgi?address=77+Massachusetts+Avenue%2C+Cambridge%2C+MA>

Since this site allows you to construct a URL to generate the latitude-longitude pair, we can use this site directly to answer our queries, and parse what is returned. Alternately, they provide the code on the site, so we could take it and use it as we please.

2.1.3 Google Maps

Google Maps was a tremendous breakthrough when it was released in February 2005. Highly interactive maps existed up to that point, but none could match Google in its lightweight JavaScript-based approach. Later on, Google published the API for its map tool. This was extremely useful, because it gave us a base framework on which to build MIT Maps and future map applications.

2.1.4 MIT Maps

Much of the functionality provided by the map applications we have created relies on the API provided by MIT Maps [2].

MIT Maps is a prototype Internet application for the visualization of highly localized spatial data. It was the product of a UROP and is the current study of an M.Eng thesis to demonstrate the workability of a robust visualization engine for federated spatial-data repositories of arbitrary size and resolution. It is modeled after Google Maps and, in fact takes advantage of some of the adaptive and asynchronous data-fetching mechanisms first introduced by Google for the web.

MIT Maps is a Web Services front-end to a more elaborate geospatial data model currently under

development. It is evolving away from the Google Maps paradigm to support more advanced data-structure handling such as direct vector drawing, topological and semantic integration, and high-grain 3D display.

The current version MIT Maps, in accordance with its design goals, has demonstrated its viability as the back-end and programming foundation for high-level, location-based applications that offer sophisticated user experiences.

2.1.5 IMS

IMS, or Instructional Management Systems, is maintained by Facilities and serves up images of the MIT Campus. It is used by services such as WhereIs [22]. This service is important because the standard Google Maps view of the MIT campus does not provide a sufficient level of granularity to show buildings and floor plans.

The IMS data exists on a number of “layers”. These layers represent different features of the MIT Campus map. There are 16 layers, as shown in Figure 2-3.

Aerial	Roads	Other+Buildings	bldg-iden-8
River	Misc	Building+Walkways	road-iden-8
Landscape	Landmarks	Buildings	green-iden-10
Greens	Parking	Rooms	landmarks-iden-10

Figure 2-3: IMS layers.

Many of these layers are self-explanatory. Some notable examples: Aerial represents the satellite image view of the MIT Campus, and the layers with “iden” in the name show labels for buildings, roads, greens, or landmarks.



Figure 2-4: Examples of “iden” layers being displayed. On the left, we can see labels for roads, buildings, and landmarks. On the right, we see labels for greens.

2.2 Building Model Generation

The Building Model Generation group, or BMG, is a team whose chief aim is to encode MIT's campus in a structured and extensible way, and to produce models of the campus in 2-D and 3-D. The inputs to the BMG pipeline are CAD DXF files that are maintained by Facilities [24]. These files encode the MIT Campus geometry. The BMG group seeks to use these to build a data model of the campus.

2.2.1 Data Model

The data model was built by the BMG group, and will be discussed in detail in Chapter 3. To briefly summarize: the BMG pipeline is being drastically reworked to take the input DXF files and output a hierarchical XML structure.

2.3 Data Sets

Numerous location-based data sets exist around the MIT campus. The majority of these data sets are self-contained and do not interact extensively or at all with other data sets. Examples of these data sets include asset data with the Property Office, equipment data with Facilities, student/employee data, class schedules data, and hazardous material data. These data sets are maintained in a structured way, though some of them are more loosely structured than others. We will briefly give background on each of the data sets. A more detailed look at the data sets will be covered in section 4.2.

It is also worth noting at this stage that throughout this thesis we reference the “MITDB” database. This is simply a MySQL database where all of the relevant information for these data sets is stored.

2.3.1 Assets

The MIT Property Office (explained in more detail in section 2.5) maintains the asset data in an Oracle database. This data includes most items purchased by MIT departments that cost more than \$500. This includes furniture, computer, laboratory, scientific and test equipment, office and service equipment [25]. This data set is less structured than others. There is a “Comment” field that often contains important information about a particular asset. Though this makes the

data flexible and is easy for a human to read, it is non-ideal for bringing into a structured framework for federating data.

2.3.2 Equipment

Equipment data is maintained by Facilities in an SAP database. This data includes items such as pumps, fans, compressors, fire extinguishers, and other items one might see around the MIT Campus but are not considered “assets” as in section 2.3.1. This data is somewhat well structured, though there are minor variations in the descriptions of equipment that can lead to confusion (e.g. there are items called “Fire Extinguisher”, “Fire Extinguisher A”, “Fire Extinguisher B”, and many other variations).

2.3.3 Schedules

The MIT Schedules Office (part of the Registrar) maintains schedules data. They are responsible for the day-to-day operation of MIT’s classrooms, and for scheduling classroom space, both for academic courses and for special events [3]. Therefore, we can obtain data from this source as to when classrooms are in use, and how many students are in each section. This information is available at the following link: <https://student.mit.edu/cgi-bin/sfresch.sh>.

2.3.4 People

The people database contains contact information for MIT faculty, staff, and students. This data is contained in the Data Warehouse. It is useful for two reasons. First, it is used to look up email addresses automatically in the Property Tool (to be explained in Chapter 6). Secondly, it is a location-based data set in itself – many people are associated with a particular office room number.

2.3.5 Hazardous Materials

MIT's Environment, Health, and Safety (EHS) Office manages the hazardous materials data for the MIT Campus. This data is contained in the MIT Data Warehouse, which runs an Oracle database. Examples include chemicals, gas cylinders, fume hoods, magnets, and radioactive materials.

2.3.6 Colloquial Information

In addition to the location-based data sets on campus, there are numerous examples of data that is not stored anywhere. Data such as bike racks, sculptures, paintings, laboratories, and other types are not stored in any structured way, but are colloquial. Many MIT students may know about the bike racks outside the Student Center, but this is not stored in a structured way. Therefore, we must keep this type of information in mind when we are devising approaches to federate data.

2.3.7 Other Data Sets

There are numerous other location-based data sets on campus. We will discuss crime logs, event feeds, and ShuttleTrack data in this section. There are other data sets as well, though they are not on the immediate horizon to be added to the set of federated data.

The MIT Police department has crime logs that categorize the crimes that occur on the MIT Campus and off-campus living groups. This data is not well structured. The location that is listed for each crime is typically an address, which would be acceptable except that the address is often incomplete. Therefore, “155 Bay State Rd” and “450 Memorial Dr” are valid entries. In this case, the first address is in Boston and the second is in Cambridge. This is a difficult issue for an automated system to resolve. If we assume we are restricted to Boston and Cambridge, and we have a unique street name, it may be possible to resolve, but there is no guarantee.

Event feeds contain data such as research talks and seminars. CSAIL’s event calendar, located at <http://www.csail.mit.edu/events/eventcalendar/calendar.php> is one such example. This data is generally well structured – either in iCalendar (a standard for calendar data exchange) or XML format. Therefore, it would be fairly easy to federate.

ShuttleTrack is a service provided to the MIT community to track the shuttles that transport people around campus. This includes the Tech Shuttle, the Northwest Shuttle, and Saferide. This system receives latitude-longitude coordinates from the vans, which are then pushed into a table. The front-end refreshes every 10 seconds, and so if there is new data in the table the updated position will be reflected on the map. Since this data is already used in a map context and is structured for doing so, bringing it to our system would be easy.

2.4 Property Office

The Property Office at MIT is responsible for managing MIT's equipment and capital assets.

“Any item that will last more than a year, is operationally complete and can be identified as stand-alone is considered equipment. Purchases from the following product categories are normally considered equipment: furniture, computer, laboratory, scientific and test equipment, office and service equipment. The difference between Capital and Minor equipment is determined by cost. Equipment at MIT is controlled by property record beginning at a cost of \$500 or greater” [25].

2.4.1 Asset-tracking

The process of tagging and tracking assets at MIT occurs on a two-year cycle. There are a number of employees in the MIT Property Office that are responsible for going to buildings on and off-campus and scanning all items with MIT barcodes. Since the people that do this have done so for a number of years and are experienced with scanning, they know how to go about scanning the items in a particular building such that few items are missed, and that every room is scanned.

After each day's scan, the scanner will go back to the office and upload the data to his local machine. He would place the barcode reader (Symbol's PDT 6800 is the one used) in its cradle, which is connected to the computer and enables a file transfer from the reader. Once this is done, that text file can be uploaded to a database called SumProp, which contains all the tagged assets at MIT.

Periodically, a report will be generated using SumProp called a PRN file. This is an exception report, containing items that have not been found in the last specified amount of time (this is a parameter the person would set at the time he runs it). He would then print out this report, and manually go through each unaccounted-for item. He would find the names of those responsible for each item, look up their email addresses using the MIT directory, and send an email containing the relevant information. This information would need to be typed manually as well, copying it from the printed report.

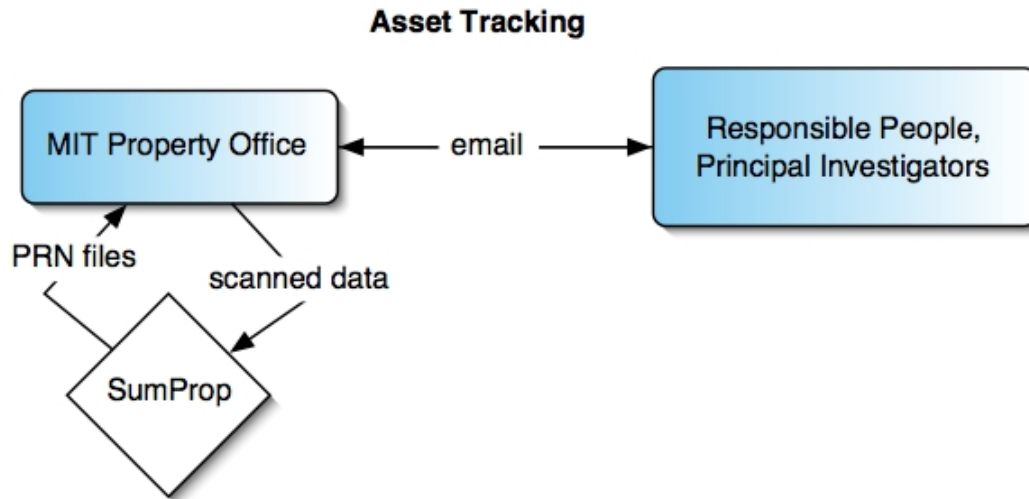


Figure 2-5: Asset-tracking by the MIT Property Office: after each day's scan, the scanned data will be uploaded to the SumProp database. Periodically, SumProp will be used to generate PRN files, which are used to send emails to responsible people and principal investigators.

Clearly, this process was one that could be improved dramatically. This improvement will be discussed in the Approach section - Chapter 4.

Chapter 3

Data Model

The BMG data model has changed tremendously over the last year, and understanding this new model is important to understanding the map applications built on top of it. We will first explore the BMG pipeline prior to the project. We will then discuss the new pipeline and the elements of the data model itself. We will also explore improvements made to augment the model, as well as what we have done to free ourselves from a dependence on IMS.

Note: The data model, both in its former and current form, has been the product of the efforts of numerous people in the BMG group.

3.1 Existing Pipeline

We will briefly describe the BMG pipeline that existed prior to the work done by the current BMG group.

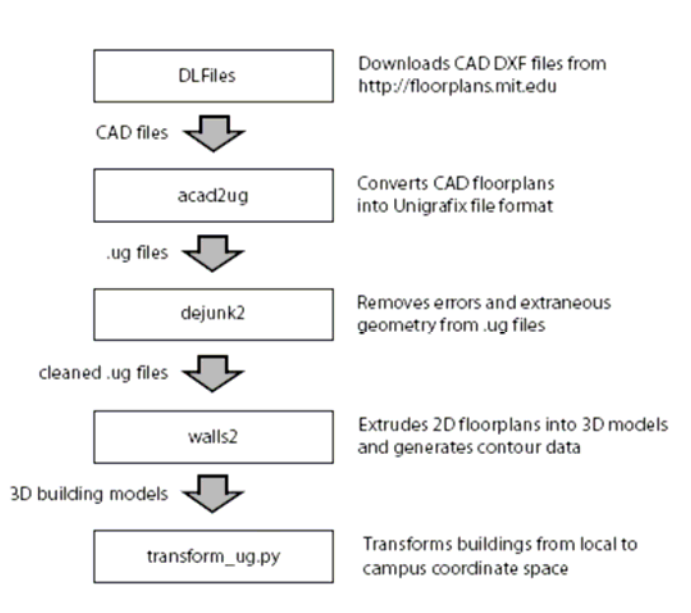


Figure 3-1: BMG pipeline overview – a set of programs download CAD DXF files from floorplans.mit.edu, remove errors, extrude the floor plans to 3D, and transform each space from local to global coordinates [26].

As shown in Figure 3-1, the MIT Campus' geometry is contained in DXF files provided by Facilities. They contain information for buildings on the MIT Campus, as well as the campus basemap. The basemap specifies building position and exterior space geometry, and allows the buildings to be aligned using the Massachusetts state plane coordinate system. These DXF files run through the pipeline and are processed, outputting a set of text files. These text files represent a graph of the spaces on the campus.

This set of text files is then read by the Location Server and represented as Java objects. This can then be used to generate routes between spaces on campus.

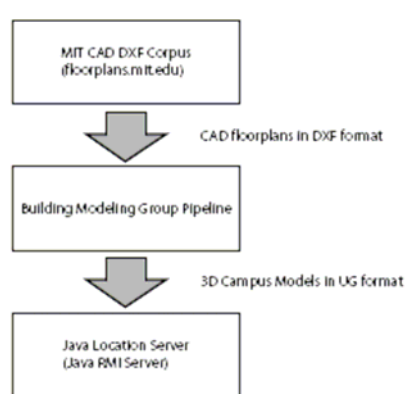


Figure 3-2: Location Server taking input from the pipeline [26].

3.2 New Pipeline

The BMG pipeline was entirely reworked in the last year. The pipeline's final output is now a set of XML files that contain the structure for the MIT Campus. Figure 3-3 demonstrates an overview of the new pipeline.

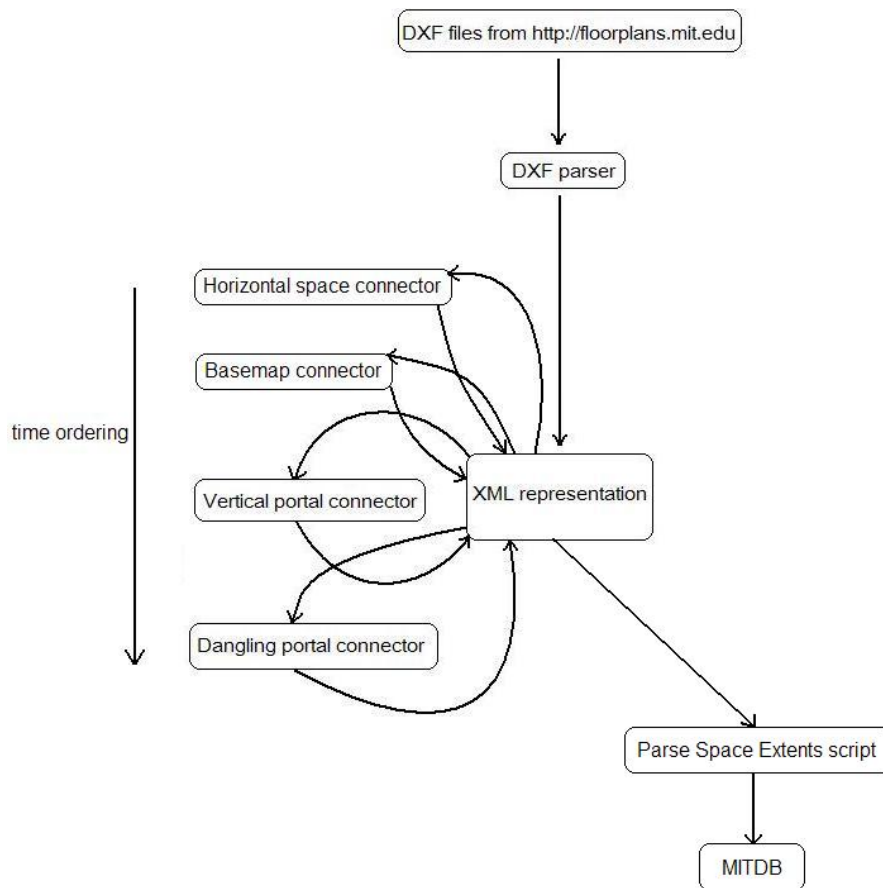


Figure 3-3: New BMG pipeline overview – DXF files are downloaded from floorplans.mit.edu, parsed and converted to XML format. The spaces are then connected and the XML representation is updated to reflect it.

After the DXFs have been parsed and the spaces converted to XML representation, four modules operate on the data to connect spaces to one another. The first is the horizontal space connector, which connects spaces within the same floor. The next is the basemap connector to connect spaces to the basemap (e.g. entrances/exits to buildings). The vertical portals are then connected (elevators and stairs). Finally, a dangling portal connector attempts to connect any remaining unlinked portals (portals the other connectors missed).

After this is finished, a script runs to parse the space extents and bring them into MITDB. This is discussed in detail in section 3.6.3.

Once the XML representation of the MIT Campus has been produced, it is then used in combination with images from IMS to form our map applications. The map applications also use the Location Server to generate routes, and MITDB for obtaining information about federated data sets. Figure 3-4 gives an overview of this process.

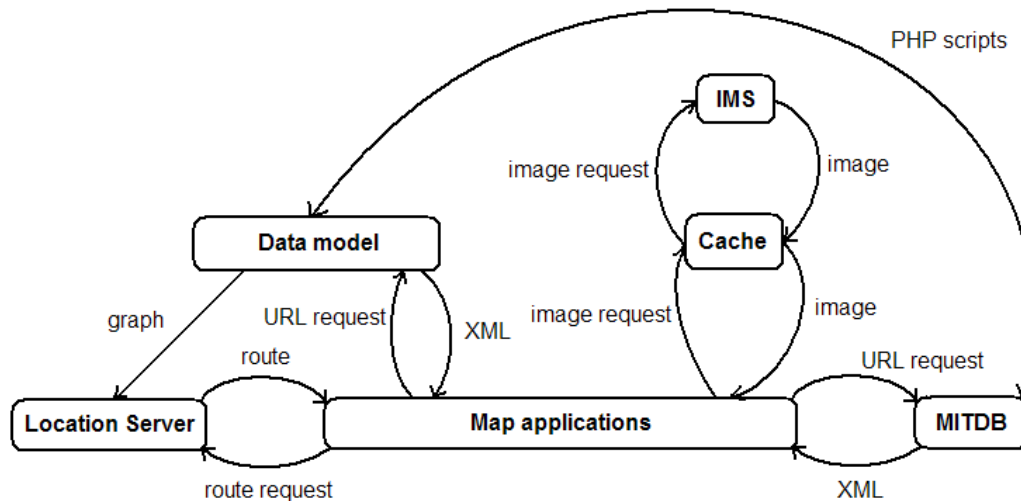
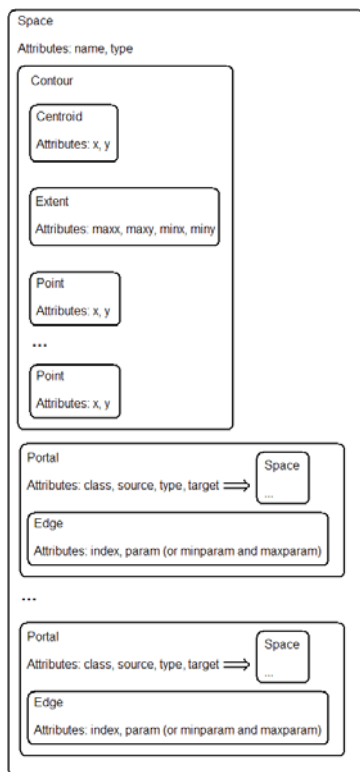


Figure 3-4: Overview of interaction between high-level modules – map applications interact with the data model, Location Server, IMS, and MITDB. MITDB and the data model also interact via PHP scripts.

3.3 Spatial Hierarchy



The two principal data structures in our model are spaces and portals. Figure 3-5 shows the structure of spaces and portals. We see that spaces contain portals, and those portals link to other spaces.

The MIT campus can be broken down into two main categories – building floor plans and outdoor spaces. Outdoor spaces are known as the basemap. Both of these are expressed in XML form, as indicated in the section 3.2. XML is a convenient way to structure geo-spatial data, as it is easy to expand to fit any new data structures we want to place in our model.

Figure 3-5: Structure overview of spaces and portals.

```

- <MITquest>
- <floor name="FLOORCONTOUR">
- <contour>
  <centroid x="709995.496398" y="495782.283739"/>
  <extent maxx="710168.874333" maxy="495890.929950" minx="709813.966149" miny="495679.630695"/>
  <point x="710165.142094" y="495826.438817"/>
  <point x="710166.052055" y="495826.853510"/>
  <point x="710136.850740" y="495890.929950"/>
  <point x="709813.966149" y="495743.782965"/>
  <point x="709843.202022" y="495679.630695"/>
  <point x="709881.647886" y="495697.151484"/>
  <point x="709883.514005" y="495693.056659"/>
  <point x="709906.490528" y="495703.527663"/>
  <point x="709908.598552" y="495698.902027"/>
  <point x="710044.486101" y="495760.829551"/>
  <point x="710040.511958" y="495769.550013"/>
  <point x="710055.298828" y="495776.288778"/>
  <point x="710064.352964" y="495756.421290"/>
  <point x="710132.979210" y="495787.696072"/>
  <point x="710123.890517" y="495807.639390"/>
  <point x="710155.132520" y="495821.877192"/>
  <point x="710158.864759" y="495813.687540"/>
  <point x="710168.874333" y="495818.249166"/>
</contour>
</floor>
+ <space name="13-1000CA" type="CORR"></space>
+ <space name="13-1000CB" type="CORR"></space>
+ <space name="13-1000E1" type="ELEV"></space>
***
</MITquest>

```

Figure 3-6: XML representation for building 13, floor 1.

We do not need to go into detail on how the basemap is structured. We will, however, look at the XML structure of the building floor plans. Each building floor plan has its own XML file in our model. These files are named “<building>-<floor>.xml”.

Each of these files contains the floor contour and the spaces within it. The floor contour is a series of X,Y coordinates that when connected, produce a polyline representing the floor plan. Each space within the floor is structured similarly to the floor itself.

3.4 Spaces

Spaces are individual rooms on the MIT campus. Each space has a name. Indoor spaces are, generally speaking, named “<building>-<floor><room number>”, though there are some exceptions to this rule (e.g. room 1-235M is building 1, floor 1M, room number 35). Each space also has a type; a particular space can be a classroom, an elevator, a corridor, a lobby, an office, or any of the 91 space types.

Similar to the floor contour, each space has a contour with a series of X,Y coordinates encoding the closed polyline representing the outline of the space. Each space also contains portals that connect it to other spaces either within or outside the floor.

3.4.1 XML Representation

As we see in Figure 3-7, each space tag has the “name” and “type” attributes. The figure shows the variety of space types found in floor 13-1.

```
- <MITquest>
+ <floor name="FLOORCONTOUR"></floor>
+ <space name="13-1000CA" type="CORR"></space>
+ <space name="13-1000E1" type="ELEV"></space>
+ <space name="13-1000LA" type="LOBBY"></space>
+ <space name="13-1000SA" type="STAIR"></space>
+ <space name="13-1006" type="LAB SV"></space>
+ <space name="13-1009" type="RCVG"></space>
+ <space name="13-1010" type="JAN CL"></space>
+ <space name="13-1014" type="M LAV"></space>
+ <space name="13-1018" type="OFF"></space>
+ <space name="13-1019" type="RS LAB"></space>
+ <space name="13-1021" type="STOR"></space>
+ <space name="13-1033" type="U/M"></space>
+ <space name="13-1035" type="OFF SV"></space>
+ <space name="13-1143" type="CLASS"></space>
+ <space name="13-1143E" type="ELEC"></space>
+ <space name="13-1143Z" type="SHAFT"></space>
+ <space name="13-1148" type="CLA SV"></space>
</MITquest>
```

Figure 3-7: XML representation for building 13, floor 1, demonstrating different space types.

If we drill down into one of the spaces (Figure 3-8), we see the contour and portals of the space. The contour has three parts within it – the centroid, the extent, and the polyline.

```
- <space name="13-1000SF" type="STAIR">
- <contour>
  <centroid x="709923.065117" y="495784.382225"/>
  <extent maxx="709929.805902" maxy="495794.868033" minx="709916.117001" miny="495771.160693"/>
  <point x="709927.870667" y="495794.868033"/>
  <point x="709916.117001" y="495789.511579"/>
  <point x="709924.479981" y="495771.160693"/>
  <point x="709928.271486" y="495772.888582"/>
  <point x="709921.843741" y="495786.992982"/>
  <point x="709929.805902" y="495790.621547"/>
  <point x="709927.870667" y="495794.868033"/>
</contour>
- <portal class="horizontal" source="true" target="13-1074" type="explicit">
  <edge index="5" param="0.500000"/>
</portal>
...
</space>
```

Figure 3-8: XML representation for space 13-1000SF, with portals hidden.

Centroid

The centroid is an X,Y coordinate representing the center of mass for the space. We could compute this on the fly from the polyline, but it is useful to have it pre-computed for each space. For example, we may use the centroid of a space to approximate whether it is within a certain distance of another space.

Extent

The extent is the bounding box for the polyline. This is also useful to have pre-computed. This will be discussed further in section 3.6.3.

Polyline

The polyline is a series of “point” XML tags. Each point contains an X,Y coordinate, and when read in sequence, the edges for the polyline are produced. This polyline is closed.

3.5 Portals

Portals are physical connections between spaces. They are either explicit or implicit. Examples of explicit portals are doors, stairwells, and elevator shafts. Implicit portals connect spaces that are connected without an explicit link between them. An example is a corridor that has two sections. The two sections have different space names, and though they are not explicitly connected with a doorway, there is nonetheless a portal between them.

3.5.1 XML Representation

As shown in Figure 3-9, portals have a class, a type, a source, and a target. The class is either “horizontal” or “vertical”. The type is either “explicit” or “implicit”. The source determines the direction of the portal. Some portals have no source because they are implicit. Finally, the target encodes the space to which the portal connects.

```

- <space name="13-1000SF" type="STAIR">
- <contour>
  <centroid x="709923.065117" y="495784.382225"/>
  <extent maxx="709929.805902" maxy="495794.868033" minx="709916.117001" miny="495771.160693"/>
  <point x="709927.870667" y="495794.868033"/>
  <point x="709916.117001" y="495789.511579"/>
  <point x="709924.479981" y="495771.160693"/>
  <point x="709928.271486" y="495772.888582"/>
  <point x="709921.843741" y="495786.992982"/>
  <point x="709929.805902" y="495790.621547"/>
  <point x="709927.870667" y="495794.868033"/>
</contour>
- <portal class="horizontal" source="true" target="13-1074" type="explicit">
  <edge index="5" param="0.500000"/>
</portal>
- <portal class="horizontal" source="false" target="13-1074" type="explicit">
  <edge index="4" param="0.285714"/>
</portal>
- <portal class="horizontal" source="true" target="13-1054" type="explicit">
  <edge index="1" param="0.338843"/>
</portal>
- <portal class="horizontal" source="false" target="13-1FLOORCONTOUR" type="explicit">
  <edge index="0" param="0.745161"/>
</portal>
- <portal class="horizontal" source="false" target="13-1FLOORCONTOUR" type="explicit">
  <edge index="0" param="0.467742"/>
</portal>
- <portal class="horizontal" target="13-1074" type="implicit">
  <edge index="3" maxparam="1.0" minparam="0.0"/>
</portal>
- <portal class="horizontal" target="13-1054" type="implicit">
  <edge index="1" maxparam="1.0" minparam="0.0"/>
</portal>
</space>

```

Figure 3-9: XML representation for space 13-1000SF, fully expanded.

Portals are two-way, meaning that they are represented twice in the data model. If we take the portal going from 13-1000SF to 13-1074, and look at the XML for space 13-1074, we will see a corresponding portal going to 13-1000SF. This structure makes sense because it allows us to explore the entire graph of spaces from a single space.

Within the “portal” tag, there is an “edge” tag. This specifies the portal’s edge along the polyline of the space. Therefore, for the portal from 13-1000SF to 13-1074, the edge index of 4 corresponds to $\{(709921.843741, 495786.992982), (709929.805902, 495790.621547)\}$.

The edge also contains either a “param” attribute, or “minparam” and “maxparam” attributes. All of these attributes are between 0 and 1 (inclusive), and represent where along the edge we are. If the “param” attribute is specified, that means we have a portal with its center at this point along the edge. A value of 0.5 means the portal goes right through the middle of the edge; at 0.75, it’s closer to the endpoint of the edge. If “minparam” and “maxparam” attributes are specified, the precise start and end points of the portal are known.

3.5.2 Dangling Portal Connector

As shown in Figure 3-9, there are portals for which the target is unknown. When the BMG pipeline is run, there are some cases where a portal is physically recognized as adjacent to a particular space, but no matching space can be found that is also adjacent to the portal. When this occurs it is called a “dangling” portal. This can occur for a number of reasons that will be explained further below.

When a portal is dangling, it is connected to the floor contour where the space is located. This is why in Figure 3-9 two of the portals have target “13-1FLOORCONTOUR”.

Dangling portals are a problem – they can potentially cut off entire sections of the MIT campus. When we are generating routes, we want to have complete connectivity within the graph of the MIT campus. In order to help achieve better connectivity, I wrote a script (`danglingPortalConnector.php` within `wikimap.csail`) that connects some of these dangling portals.

This script queries the “rooms” table of our MITDB database to get the distinct buildings and floors in descending order. Each floor’s XML file is called, and the script loops through it to find the dangling portals. When a dangling portal is found, the script does a proximity query (located at <http://agenda.csail.mit.edu:8080/proximity?name=54-913A&distance=50&units=feet> – replace the name with the desired space name) for the space containing the portal. This proximity query searches the graph for all spaces within x feet/meters of the original space. It limits results to only those floors within one level of the original space. It is necessary to allow for floors within one level because some buildings connect to each other at different floor numbers (e.g. building 26 and 16).

Once the proximity search is run, we get back a number of candidate spaces, some of which have dangling portals. These candidate portals are then analyzed to determine whether any of them are close enough matches to be “stitched” together with the original dangling portal. Six comparisons are done. The first is a slope comparison. The slopes of the portals being compared must be within 0.1 of each other. If this is met, then we determine whether the portals are close enough to be considered a match.

Originally, in order to test the shortest distance between two line segments (portals), we were going to use an algorithm that determined this. This is explained in Garrett Hoffman’s paper [15]. However, after considering it further, we determined this added an unnecessary level of complication, and we did not need such a fine-grained distance between the two segments. Therefore, we simply take the midpoint-to-midpoint, startpoint-to-endpoint, startpoint-to-

startpoint, startpoint-to-endpoint, endpoint-to-startpoint, and endpoint-to-endpoint distances, determining the minimum among them. This gives us a crude measurement of the distance between the segments. If this minimum is less than $\sqrt{10}$, we determine that the segments are close enough to be considered a match.



Figure 3-10: Connecting dangling portals between 10-631 and 10-637.

We see in Figure 3-10 why it is useful to measure the startpoint-to-startpoint and endpoint-to-endpoint distances in addition to midpoint-to-midpoint. This portal would not be found if just the midpoint distance were used.

Once a particular match is found, we add the match to our database. This is done so that we do not match the same dangling portal to multiple spaces. This is especially important at places where buildings connect.

In running this script and analyzing the results, we have found there are four main categories that all dangling portals fall into:

- 1) Portals that connect to the basemap.
- 2) Portals that leave a building, but has no corresponding dangling portal in the building to which it connects.
- 3) Portals that go to a building for which we do not have complete data.
- 4) Portals for which a corresponding portal in the opposite direction is found.

This dangling portal connector successfully handles the fourth case. The first three categories need to be addressed earlier in the pipeline in order to be successfully resolved.



Figure 3-11: Example of portals connecting to the basemap at the entrance to 77 Massachusetts Avenue.



Figure 3-12: Dangling portals from building 10 going to building 13, but no corresponding dangling portal to which a match can be made.

3.6 Federation From IMS

One of the goals in creating our data model is to create a structure that is free from a dependence on the IMS server. We've done this in three main ways: contour drawing, image caching, and point queries.

3.6.1 Contour Drawing

Previously, the IMS server produced all space contours. Therefore, if we wanted to get the contour for building 10, we would see the image in Figure 3-13. We could similarly do this for individual rooms. The image produced would have the desired space highlighted in red. This worked well if we wanted just one or two spaces highlighted, but it quickly became unwieldy for



Figure 3-13: IMS-returned image for building 10.

multiple spaces. In fact, because of 256 character limits in the URL parameters (for a GET request), it was impossible to ask IMS to produce more than 40 space contours. IMS would crash long before 40 spaces, however. Once IMS is asked for 10+ spaces, it begins to struggle and will eventually crash. Therefore, IMS is non-ideal for generating multiple space contours.

Instead of using this method for drawing contours, we use a new method, demonstrated in Figures 3-10, 3-11, and 3-12. This is done using the GPolyline constructor as provided by Google Maps. The polyline encoded in the space's XML is used to produce the correct contour.

3.6.2 Image Caching

With the contour drawing free from IMS, the next step was to free our map applications from IMS images. Originally in MIT Maps, each tile in the map was produced by a query to IMS. These images were then generated by IMS and pushed back to be displayed. This process was slow, and led to a poor user experience. On top of that, facilities began complaining because we were pelting their server too many times.

In order to address these issues, we implemented a caching mechanism. For all image tiles within the first 9 zoom levels, we first query our server to see whether we have the image stored. If we do not have it stored, we query IMS to get the image and save it to our server. Therefore, each tile within the first 9 zoom levels will only result in one IMS query.

The naming scheme we use is quite simple. At the highest zoom level, there are two tiles – therefore these tiles are named “1” and “2”.



Figure 3-14: MIT Campus at the highest zoom level.

As we lower the zoom level, each of these tiles is broken into four pieces. Therefore, at the 2nd highest zoom level, there are 8 tiles. At the next, there are 32, then 128, and so on. As we drill down into lower zoom levels, we simply keep adding to the name. So if we want the lower left quadrant of the “2” tile, we can call it by asking for “23”. If we’d then like the upper left quadrant, we ask for “231”, and so on.

1	2
3	4

Figure 3-15: Each tile has four tiles at the next zoom level.

This is an easy way to drill down into the map. Figure 3-16 shows how we can find an image of building 2 by starting from the highest zoom level and determining which quadrants to zoom to.



Figure 3-16: Starting with image “2”, we add to the image name until we reach “231232”.

This naming scheme is also convenient because if we take the length of the image name, we have the zoom level.

We store three different layers on our server – map, aerial, and hybrid. The image names end in “m” for the map layer, “a” for aerial, and “h” for hybrid. Images are stored in a directory structure, where each number in the name is a folder (except for the last number). Therefore, the map layer for “231232” would actually be located at /2/3/1/2/3/2m.jpg. This allows for a very scalable structure for storing and retrieving the cached images.



Figure 3-17: Three layers of the same “231232” image tile – type “m” (left), type “a” (middle), type “h” (right)

The code for this caching mechanism runs in a single script contained at:

<http://wikimap.csail.mit.edu/cacheImages.php>

Any time a map application would like an image tile, this link is accessed with the appropriate parameters (“?img=231232&type=m” for the left image of Figure 3-17). If it is within the first 9 zoom levels, the image will be stored in our cache. Otherwise, it will be thrown away. We made this design decision because the high granularity zoom levels are infrequently used. Browsers often do image caching as well, and so this can be used to aid a user that is continually requesting high granularity images.

In the process of image caching, we wanted to achieve atomicity. We were finding that without this assurance, cached images would be incomplete when the script timed out. In order to fix this problem, we create a dummy file called “CF<filename>” in the same directory as the cached image. This is done at the very end of the script, so if this file is there when we’re searching for a particular image, we can be assured the image was correctly cached.

As mentioned previously, we only cache the first 9 zoom levels of the MIT Campus. This is due to the exponential nature of these tiles. At the lowest zoom level (there are 15), there are $4^{15} / 2 = 5.4 \times 10^8$ tiles. The images range in size from approximately 5KB to 100KB, with the average being somewhere around 40K. If we use this figure, we can see that storing the last zoom level would require approximately 29 terabytes of disk space per layer. If we store up to the ninth zoom level, each layer would use about 7 gigabytes. The 10th zoom level would add another 21 gigabytes per layer, so we decided to stop at nine.

3.6.3 Point Queries

The final dependence we have on IMS is for point queries. A query of this type asks for all locations that contain a certain X,Y coordinate. IMS can return matches to point queries for a number of layers – the ones that are important here are the Rooms, Greens, Landmarks, and Parking layers. Given an arbitrary X,Y coordinate on the MIT Campus, we would query IMS to ask whether each of these layers contains any matches for this coordinate.

The problem with doing this, however, is that we discovered we could not rely on IMS to provide this service. In fact, IMS entirely removed point query functionality on the Rooms layer. After this occurred, we needed to provide this functionality ourselves. The data model we put in place made this fairly easy to do. Since each space has a bounding box extent, we can determine which spaces have bounding boxes containing the point. For better precision, we can then do a point-in-polygon test (see Appendix D.1). The script that does this point query is called “pointQuery.php”.

The problem with this is that to search through the entire structure of our data model would be slow. In order to speed up the process, we have a script (parseSpaceExtents.php in the ParseFunctions folder of wikimap.csail) that loops through all the spaces in the data model and inserts the space extents into the “rooms” table in our MITDB database. This script runs at the end of the BMG pipeline. This is convenient because if each of the four columns representing the extent are indexed, we can determine the spaces that contain a particular point in $O(\log N)$ time, where N is the number of spaces in the data model [7]. Making this change has led to significant speed improvements with the point queries in comparison to IMS.

We still rely on IMS for point queries on the other three layers, however. This is because we currently do not have the greens, landmarks, and parking lots encoded as spaces in our data model.

Chapter 4

Approach

This chapter will discuss the approach used to federate a number of distinct data sets for use in a map context, as well as the approach to the applications created to visualize this data.

4.1 Overall Approach

Early on in the evolution of this project, the MIT WikiMap and the Property Tool were separate entities that did not fully take advantage of each other's functionality. Since one of the goals of this project was to create a framework for federating location-based data sets to bring them into a map context, it would not be sufficient to simply have two separate applications.

Therefore, it is critical that our approach takes into consideration a wide-range of potential location-based data sets. When we seek to make generalizations about the data sets, we must be sure they are applicable to other data sets and problem spaces; if a solution were developed only considering the MIT WikiMap and the Property Tool, that would likely not be extensible. As this is done, we must carefully document the methods that will be used, such that this can later be included in an API of the system.

Generally, in federating data, it must be possible to get the most basic information from the data provided – namely, the location. This location may be specified in a variety of ways, and the federation tools should be adaptable based on this. As mentioned in section 2.1.2, the types of input supported should include:

- Massachusetts state plane coordinates
- Latitude-longitude (GPS) coordinates
- MIT buildings, rooms, landmarks, and colloquial names
- Street addresses

As will be specified in the API, the location format must be provided. Though the format of the location should be flexible, this information is required. Putting data in a map context without location information would not serve much purpose.

Aside from this, the federation tools should be able to map field names in the original data to commonly known field names such as “Item Name”, “Category”, and others. However, it should also be flexible, allowing an administrator of a data set to include any type of field. In other words, we should not make the unnecessary restriction of fitting the data to set fields, because we can potentially lose valuable information this way.

At the most basic level, the federation mechanism will simply be a URL that data administrators can use to add to the system. The query string in the URL contains the information necessary to bind data to the necessary fields. This can happen in a variety of ways. The URL can be for one piece of data at a time, such as with the MIT WikiMap:

```
"http://wikimap.csail.mit.edu/wikisaveobjproc.php?email=" + useremail + "&nonce=" + nonce +  
"&name=" + itemNameE + "&location=" + itemLocationE + "&gpointx=" + itemGPointX +  
"&gpointy=" + itemGPointY + "&category=" + itemCategoryE + "&image=" + itemImagesE +  
"&description=" + itemDescriptionE
```

The URL's fields correspond to the necessary data fields. This method can work for individual cases, where we tailor to the particular data set, or for cases where we'd like to more strictly control how data is entered into the system.

However, we need a more general mechanism for data sets that do not fit into this paradigm. This mechanism may be as simple as saying that for fields that are unknown to our system, it will lump all of them into one chunk of data. It could also be a mechanism where an administrator provides mappings between the fields in the system and fields in his data (and putting some kind of “flexible” field tag for field tags that do not fit otherwise).

The mechanism may also be for more than one piece of data at a time. It may be feasible to provide a federation tool for entire databases. This tool would allow an administrator to provide the necessary details about the database, and a mapping between the fields in that database and the required fields in our system. A script provided by this system could then be run to scrape all the data from the database.

The infrastructure put in place must be able to support all the existing tools that have been built and the functionality desired in those applications. At the same time, however, there must be a programmatic infrastructure in place for people to easily federate data. In the end, we should be able to demonstrate the infrastructure with a number of example data sets, and create visualizations using the structure we have specified.

In terms of the interface to be used after data has been federated, this system should have a number of properties to make it robust, flexible, and easy-to-use. These properties will be discussed.

Web-based:

Our infrastructure should support easy, rapid construction of web applications that run in vanilla browsers, with the only requirement that JavaScript be enabled in the browser.

Coordinate-neutral:

As indicated earlier, we should support a useful variety of coordinate systems, and provide conversions between them. The set of four include: 1) Massachusetts state plane coordinates 2) GPS coordinates 3) MIT buildings, rooms, landmarks, and colloquial names 4) Street addresses. Our map applications will use Massachusetts state plane as the default to which all conversions are made.

Automatic caching:

The system should also be structured in a manner that enables caching of outside data. If possible, the system should fetch the real-time data, but if that cannot be done, there should be a backup version to load from. We would like to have a suite of scripts that can run automatically and populate the cache with the most recent data. We would also like a system in place to propagate changes made to our version of the data. If the cached version changed and nothing was propagated, then the real-time data would eventually overwrite the changed version of the cache. This is not the behavior we would like.

Real-time:

The capability for feeding in real-time data should be possible. Examples on the MIT campus where this is relevant are with ShuttleTrack data and MIT events RSS feeds – if we wanted to incorporate this into the system, the real-time capability is necessary.

Cleanly deals with scale issues:

When number of items displayed is large, there must be a system in place for removing clutter from the map. One of these methods was developed for the MIT WikiMap. This method determined the extent of the viewable part of the map, then looped over the items currently displayed. If two items were found to be too close to one another (based on the extent), one of them would be removed from the map. If the user zoomed in, at a certain point the extent would be small enough such that both items would display.

Easy to create visualizations of data:

Visualizations of data may be discrete, such as placing icons on the map to indicate the locations of various objects. The objects that display should be customizable; the text that appears with these objects should also be customizable.

Visualizations may also be continuous (at least as it appears to the user), such as a colorization of the MIT campus to indicate the price of assets per square foot. The way this may work is to color using concentric circles. To give an example, assume that there are a number of assets in 10-250 and none in the adjacent rooms. The center of 10-250 would therefore be a bright red color. As you got farther away from that point, the area of the circle with center at 10-250 would get bigger, and so the number of assets per square foot would decrease proportionately. Coloring in this way, with a number of transparent layers, will lead to visualizations that appear to be continuous.

Easy to combine information from multiple data sets:

For analysis purposes this is important. For example, we might like to calculate evacuation routes for students in classrooms in the event of a hazardous material problem.

Queries and visualizations should be easily repeatable:

We should be able to easily repeat any query or visualization made. We can do this by encoding in the URL the necessary parameters to construct the visualization from scratch. In the WikiMap we do this with a “permalink”, which once clicked places the necessary parameters in the URL. We can also achieve this by tying unique nonces to particular displays, as will be demonstrated in the Property Tool.

Seamless embedding of mouse and keyboard handlers:

It should be possible to drag objects around in this system and drop them on desired locations. It should also be possible to attach any intuitive action we might need to mouse and keyboard events. It should be possible to attach informative hooks to these events. For example, when a user mouses over an MIT WikiMap icon, the contour for its containing space will appear.

Flexible and persistent tagging of locations and objects:

Our system should make it easy to attach tags to any location. This tag will persist and be immediately available upon searching.

Unique naming system:

All elements of interest must have a unique name in this system, even as the number of data sets we have federated gets large.

Easily searchable:

All data sets that are federated should be searchable, and the hooks into these data sets should be modular and easy to extend.

4.2 Data Representation

All objects in the system must have some common structure; otherwise this system would not be easily extensible. As mentioned in section 4.1, all we really need for each object is the location. On top of this, however, we also need a way to refer to each object. Therefore, each object must have some unique ID such that even as the number of data sets is large, all objects can coexist without collisions. Figure 4-1 demonstrates a simplified data representation for all objects.

Unique ID
Location
Location format
GPointX
GPointY
Other Fields

Figure 4-1: Simplified data representation for all objects.

The reason we have Location, GPointX, and GPointY is because even once we have a location, we may not necessarily be able to place it in the system. Therefore, if a user added an object to the MIT WikiMap and gave it a non-standard location (e.g. “pika” for an off-campus living group), our system would have no way of resolving the location provided, and would need to rely on where the user placed the object on the map in order to bind the location to that particular place. Since the map’s default coordinate system is Massachusetts state plane, the GPoint (x,y state plane coordinates) for where the object was placed is then stored.

“Other Fields” is part of Figure 4-1 because we have the ability to add an arbitrary number of fields to each object. These fields can be named as the user desires; so long as the name doesn’t conflict with one of our other names (“id”, “location”, “gpointx”, and “gpointy”), it can be used.

This data representation is contained in the back-end. When a request is made of the back-end, each data object that is returned will have this information at a minimum.

4.3 Data Federation

The majority of the location-based data sets on the MIT Campus are self-contained and do not interact extensively or at all with other data sets. Examples of these data sets include asset data with the Property Office, equipment data with Facilities, class schedules data with the Registrar, people data, and hazardous materials data. These data sets are maintained in a structured way, though some are more loosely structured than others.

Due to the variety of location-based data sources, we must be flexible in terms of the ways we access them. It is not always a simple matter of getting a username and password set up to access the relevant database. We must sometimes need to work around access restrictions and other roadblocks that arise. Generally, we have employed four methods for federating data:

- 1) Running a URL on our system to federate a single piece of data
- 2) Accessing an outside URL to scrape data from it
- 3) Parsing a file provided by a data administrator
- 4) Directly accessing a database with a username and password

When we use one of these methods to federate a data set, we bring it into our database – MITDB. We copy all fields and values provided to us, as we do not want to throw away data. In addition, we update our “datamapping” table so that we know all the pertinent information needed to populate the minimal data structure shown in Figure 4-1. Thus, we must provide the table name of the data set, the field(s) for the unique ID, the location field(s), and the location format.

We will now discuss how each of the data sets are structured, and how they are accessed.

4.3.1 Assets

The asset data is mostly well structured, and is contained in a single table in an Oracle database. The fields in the table are shown in Figure 4-2.

TagNum	SerialNum	UseStatus	TagSeq	Title	StatusDate	Life	AcqAmount
Flag	Manufacturer	Disposition	DeptNum	UserDeptNum	DispCaseNum	FollowOnContract	GovTag
ID	Supplier	Account	ChargedDept	Department	DispDate	SponsorType2	
Location	ObjCode	PI	PayDate	Building	DispSalesPrice	Title2	
RespPerson	AcqDocNum	SponsorType	Units	Floor	GovDispDoc	Tagged	
TagStatus	AcqDocType	Sponsor	TransDate	SBldg	Class	LastInvDate	
StandardName	AcqMethod	ContractNum	Entered	SFlr	Comment	PayYr	
Model	AcqDate	Amount	AcctSuffix	CaseNumber	Trns	Tagger	

Figure 4-2: Asset fields.

The important fields here are Location, TagNum, StandardName, PI, and RespPerson. Location is most important, because without that piece of data, it is impossible to place assets in a map context. TagNum is important because it uniquely identifies an asset. StandardName is simply the asset's name, such as "Laptop" or "Scanner". PI and RespPerson are important because these are the contact persons for that asset. When the Property Office is doing follow-up, these are the people that are contacted first (typically RespPerson before PI).

The original plan was to get direct access to the Oracle database in order to improve the end-to-end functionality of the Property Tool. However, due to administrative red tape, we are still restricted to federation mechanism #3 – parsing a file provided by a data administrator. As described in section 2.4.1 and Figure 2-5, SumProp can be used to generate PRN files containing all the relevant information for a particular item. These PRN files can be parsed by one of the suite of scripts we have for data federation (in this case, `parseDelimited.php?type=comma` in our ParseFunctions folder of `wikimap.csail`), and corresponding entries will be created in the "assets" table of our MITDB database.

4.3.2 Equipment

Equipment data is well structured, but because it is within an SAP database it has a fairly complex structure. In the SAP database (maintained by Facilities) where the equipment data resides, there are a number of internal fields that make getting at the data directly difficult. In fact, Facilities indicated that they could not allow a program to access their data directly. Therefore, similar to the asset data approach, we needed to parse reports provided by Facilities. The report provided contained the fields shown in Figure 4-3.

Acquisition date	Changed on	ConstructYear	Loc/AccAssmt	Long text
Main WorkCtr	Serial number	Sort field	System status	Object type
TS changed on	TS created by	TS created on	ConsecNumber	Valid from
Valid to	Work center	Equipment	Description	FunctLocation
Room	Manufacturer	Model number	SuperEquip.	Location
ManufSerialNo.	ManufPartNo.			

Figure 4-3: Equipment fields.

The most important fields here are Location, Room, Loc/AccAssmt, and Object type. The full location is actually spread across two fields in this data set. Therefore, we must concatenate Location with Room (<Location>-<Room>) in order to successfully resolve the location. Loc/AccAssmt is similar to TagNum for assets, in that it is a unique identifier for a piece of

equipment. Object type is similar to StandardName for assets, and provides a categorization for each piece of equipment.

In order to interpret the report given to us by Facilities, we take the Excel file they provide to us, save it as a comma-delimited file, and then use the `parseDelimited.php?type=comma` script to parse it.

4.3.3 Schedules

The schedules data is very simple in its structure, as there are not many fields. The most important are Location, Subject, and Day/Time.

Term	Subject	Section	Location	Day/Time
Students/ Section	Students/Subject	Students/Cluster	Num of Sections	


Figure 4-4: Schedules fields.

Location is the most important field, and is an MIT room number.

Note: Students/Cluster represents everyone in the course (for courses with multiple numbers). Students/Section represents students registered under each course number. Students/Section is a count of the students in that section (e.g. a recitation for a much larger overall course).

This data is held by the Registrar, and can be attained in one of two ways – either by accessing a website to query for the desired class schedules (see Figure 4-5), or by looking at a website containing all the class schedules for the current term in tab-delimited form:

<http://web.mit.edu/registrar/www/schedules/csbindex.shtml>



Subject Schedules By Section

This creates a report or download file of subject schedules, listing all sections for each subject. The list may include all subjects or may be limited by the data items below.

[Help](#)

Create A New Report			
First (or only) Term:	<input type="text"/> (required)	Last Term:	<input type="text"/>
Second Term Range:	<input type="text"/> (optional)	Last Term:	<input type="text"/>
Select Bldg Room:	<input type="text"/>	Or specify:	<input type="text"/> (i.e. 1012 for a building, or 1012.200 for a room)
Select Subject Dept:	<input type="text"/>	Or specify:	<input type="text"/> (i.e. 1012 for a department, or 1012.01 for a subject, note the period.)
Time Period:	<input type="radio"/> 1st week <input checked="" type="radio"/> 5th week <input type="radio"/> End of term		
Type of class:	<input checked="" type="radio"/> All <input type="radio"/> Joint or "Meets with" <input type="radio"/> Joint Discrepancies		
Type of section:	<input type="checkbox"/> Design <input type="checkbox"/> Laboratory <input type="checkbox"/> Lecture <input type="checkbox"/> Preparatory work for lecture <input type="checkbox"/> Recitation <small>(you may select more than one)</small>		
Descriptive Label:	<input type="text"/> (required for email notification)	Email Address:	<input type="text"/> (required for email notification)

[Help](#)

View the report from this page:

Create a download file; stay on this page until the file is ready:

Create a download file; be notified by email when the file is ready:

Figure 4-5: Web interface for obtain class schedules – <https://student.mit.edu/cgi-bin/sfresch.sh>

Currently the data is federated by running a script on our server (getSchedule.php in our WebScrape folder of wikimap.csail) that scrapes data from the site containing the schedules for the current term. One reason for doing this is because it is the only data set that currently requires web scraping – because it is functionality we would like to have for our system, it was decided to implement it here.

A couple of notes to make about this data – though the individual fields are relatively easy to federate, the fields need to be further interpreted. For example, “Section” will be:

- 1) Lecture Section = 'L01', 'L02', etc.
- 2) Recitation Section = 'R01', 'R02', etc.
- 3) Final Examination = 'Q01'
- 4) Comment = '01', '02', etc.

This is not difficult to interpret, as there is a limited set of possibilities. A more difficult field to interpret is the “Day/Time” field – this can be ‘MWF1’, ‘MW8.30-10’, ‘MTWR EVE (5-7.30 PM)’, ‘W1-5’, ‘TR8-9.30’, or any number of possibilities. In order to interpret this, we needed to create a custom parser. This parser does the following:

- 1) M - Monday, T - Tuesday, W - Wednesday, R - Thursday, F - Friday
- 2) No dash means a one-hour class.
- 3) 8 to 11:30 are AM, 12 to 7:30 are PM, unless otherwise noted.

For #3, we needed to make an assumption for entries that had no AM/PM label. Since MIT classes do not start before 8 AM, this assumption is a safe one. If this were to change in the future, we could easily change the parsing rules to account for it.

Using this field parsing, we can easily interpret each class schedule. This web scraper/interpreter can be run as often as we like. Class schedules do not change much after the first couple of weeks each semester, so it can be run sparingly after that period.

4.3.4 People

There are two flavors of people data – students and employees. They are contained in the MIT Data Warehouse under the table names “MIT_STUDENT_DIRECTORY” and “EMPLOYEE_DIRECTORY”. Figures 4-6 and 4-7 show the fields for each table.

Mit Id	Office Location	Department Name
Last Name	Office Phone	Krb Name
First Name	Directory Title	Krb Name Uppercase
Middle Name	Primary Title	Email Address
Full Name	Department Number	Name Known By

Figure 4-6: Employee fields.

Last Name	Office Location
First Name	Office Phone
Middle Name	Email Address
Full Name	Department

Figure 4-7: Student fields.

Office Location is the only location-based field. Email Address is also important, as it is used in the Property Tool to retrieve contact information.

Since this data is contained in the Data Warehouse, it is possible to use federation mechanism #4. A username and password was set up for accessing it. However, despite this direct database connection, the only tool currently available for using the database connection was BrioQuery. According the Data Warehouse website:

“Data from the Data Warehouse is brought to your desktop by a query tool; BrioQuery is the software most often used at MIT. Once the data is stored locally (in your computer) you may format it to your liking and create a myriad of different reports. Or, you can export the results of your query into other software programs, such as Microsoft Excel, and manipulate the data

locally. Your results may also be joined with local information from your own files. This is very useful for instances when you need to create custom reports with information that is not stored in any of MIT's current systems” [24].

We could certainly set up a bridge to access the data directly, and the Data Warehouse has done this before. In fact, it is necessary for the Warehouse to do this in order to do its nightly updates from the real-time systems. However, because people data does not change frequently, it was determined that this step was not necessary currently. Our infrastructure allows us to easily change federation mechanisms, so this could be part of future work to be done. For now, however, we are essentially using federation mechanism #3 – using BrioQuery to export a file, then parsing this file to get all the information we need (parseDelimited.php?type=comma).

It is also possible to access the data by using mechanism #2. As shown in Figure 4-8, we can construct a URL that returns information for a particular person. This information can then be parsed. We can even potentially do this on the fly – meaning, if a user asks for a particular person in our system, we query this URL, parse the data, and return it.

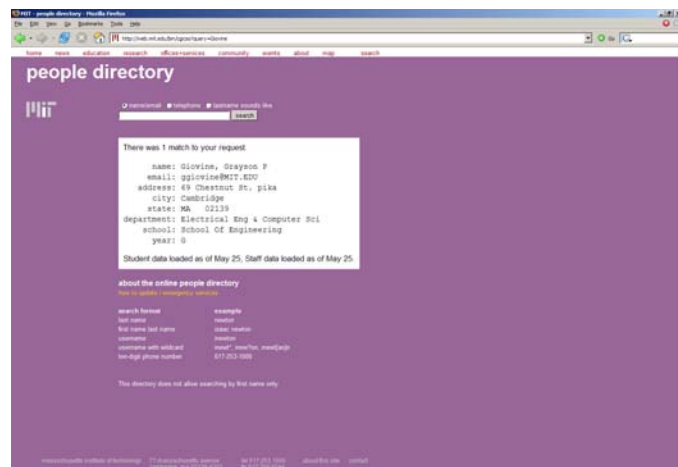


Figure 4-8: Query for “Giovine” on MIT website.

4.3.5 Hazardous Materials

Hazardous materials data is also contained in the Data Warehouse. There are numerous tables, though the most important one for our purposes is “EHSS_HAZARD_SARA_COMBO”, which essentially combines a number of tables into one. Figure 4-9 demonstrates these fields.

Ehss Room Key	Ehss Hazard Key	Hazard Name	Ehss Sara Substance Key	Sara Substance Name	Is Hazard
Is Sara	Is Room Inspection Req	Room Type Code	Room Building Number	Room Name	Room Description
Room Code	Room Floor	Room Usage	Room Area	Is Subroom	Is Shared Room
Room Set Name	Is Sara Relevant Room Set	Dlc Name	School Area Name	Pi Mit Id	Pi Personnel Number
Pi Full Name	Pi Title	Pi Email	Pi Office Loc	Pi Office Phone	Pi Alt Office Loc
Pi Alt Office Phone	Pi Home Phone	Pi Emergency Phone	Ehs Rep Mit Id	Ehs Rep Personnel Number	Ehs Rep Full Name
Ehs Rep Email	Ehs Rep Office Loc	Ehs Rep Office Phone	Ehs Rep Alt Office Loc	Ehs Rep Alt Office Phone	Ehs Rep Home Phone
Ehs Rep Emerg Phone	Sara Reporter Mit Id	Sara Reporter Personnel Number	Sara Reporter Full Name	Sara Reporter Email	Sara Reporter Office Loc
Sara Reporter Office Phone	Sara Reporter Alt Office Loc	Sara Reporter Alt Office Phone	Sara Reporter Home Phone	Sara Reporter Emerg Phone	Room Status
Room Set Status	Dlc Code	Hazard Quantity	Sara Valid From Date	Sara Valid To Date	Sara Quantity
Sara Unit Of Measure	Sara Quantity In Pounds	Sara Object Status	Sara Status Code	Sara Status Description	Sara Exposure Rating Category
Sara Last Change Date	Sara Last Change User	Sara Risk Assessment Id	Sara Report Year	Record Counter	Warehouse Load Date
Sara Report Submission Status					

Figure 4-9: Hazardous materials fields.

There are too many fields to detail all of them here; however, the most important field for a map context is “Room Code”. “Room Name” also contains location data, though it also has extraneous information from a location standpoint (e.g. “16-752-SHARED” to represent a shared room between labs; “16-752” will be the entry for “Room Code” here).

The federation mechanism for hazardous materials is similar to that of people data – a combination of mechanism #3 and 4. Therefore, we use BrioQuery to download all the fields in the table, export to a file, and parse it using parseDelimited.php.

4.3.6 Colloquial Information

As indicated in Section 4.1, we federated colloquial information using mechanism #1 - running a URL on our system to federate a single piece of data. Since objects are typically added to the WikiMap individually, it made sense to do this. Any time a user adds or edits an object in the MIT WikiMap, a URL will be asynchronously fetched, and this will update the MITDB database.

4.3.7 Other Data Sets

For crime data, it would be ideal to access the police database directly. However, due to privacy concerns raised by employees with MIT Police, this is simply not possible. Therefore, the best we can hope for is mechanism #3. A data administrator could provide a limited set of fields for crimes occurring around the MIT Campus, and we can parse it. The fields they provided are shown in Figure 4-10.

Date & Time Reported	Inc Type	Date & Time Occurred
Address	Comments	Disposition

Figure 4-10: Crime log fields.

The most important field is “Address”, but this field is difficult to interpret, as explained in section 2.3.7. There may be an entry such as “AEP / 155 BAY STATE RD” or “W61 / 450 MEMORIAL DR #G-312”. Though it is relatively easy to interpret if the address contains an MIT building number, it is more difficult when there is an off-campus living group or fraternity. In order to interpret these, we will likely need bindings provided either from another data source, or from the MIT community.

ShuttleTrack data is contained in a database and is already accessed through a web interface. Therefore, bringing this to our system would be fairly easy. Since our code is structured in a way that facilitates easily changing the type of database being accessed, nothing in the infrastructure would need to change.

MIT Event feeds would also be easy to federate, especially if we use RSS feeds. This would be a simple matter of parsing XML, which we already use heavily in our infrastructure. If we were using iCalendar instead, we could use an iCalendar parser to interpret the data.

4.4 Space Interpretation

In addition to federating a number of location-based data sets, it was also necessary to parse and interpret all MIT room numbers and bring them into MITDB. Generally speaking, MIT rooms are named <building>-<floor><room>, but there are exceptions and complications. For example, mezzanine floors have an “M” at the end. Therefore, a room such as 35-528M should be interpreted as building 35, floor 5M, room 28.

Secondly, room numbers have at least two characters, but sometimes more. This would not be a problem to interpret except that there are some floors on campus that have two digits. Therefore, how do we distinguish between 54-1131 and 1-015E? Luckily, almost all rooms obey the convention that there are two numerical characters at the start of the room number, followed by either non-numerical characters or nothing. When there is a room that does not obey that convention, there is a slash – so in the case of 13-1010J, it is provided to us as “13-1/010J”, which makes it clear how to interpret it.

Finally, building 32 (Stata Center) does not obey common conventions in that there is a non-numeric character at the start of the room number. Furthermore, the non-numeric character

comes immediately after the dash. Therefore, 32-D400CA should be interpreted as building 32, floor 4, room D00CA.

With a few simple rules, we can interpret and structure all formal space names on the MIT Campus. The script that runs is called `populateSpatialData.php` and resides in the `ParseFunctions` directory of `wikimap.csail`.

4.5 MIT WikiMap

Our approach to the MIT WikiMap took into account both a desire for flexibility as well as a desire for some structure. The principal goal in its design was to capture the wealth of colloquial information on the MIT Campus. There are two main ways to approach this problem. The first is a more unstructured approach, allowing users to create their own categories and to place objects in the system as they choose. The second is to impose more structure on the system, and have a fixed set of categories that can be added to the infrastructure. We chose to go with the second option for the sake of consistency. Too much freedom would create a more confusing user interface, and would lead to redundant groupings such as “architecture” and “buildings”.

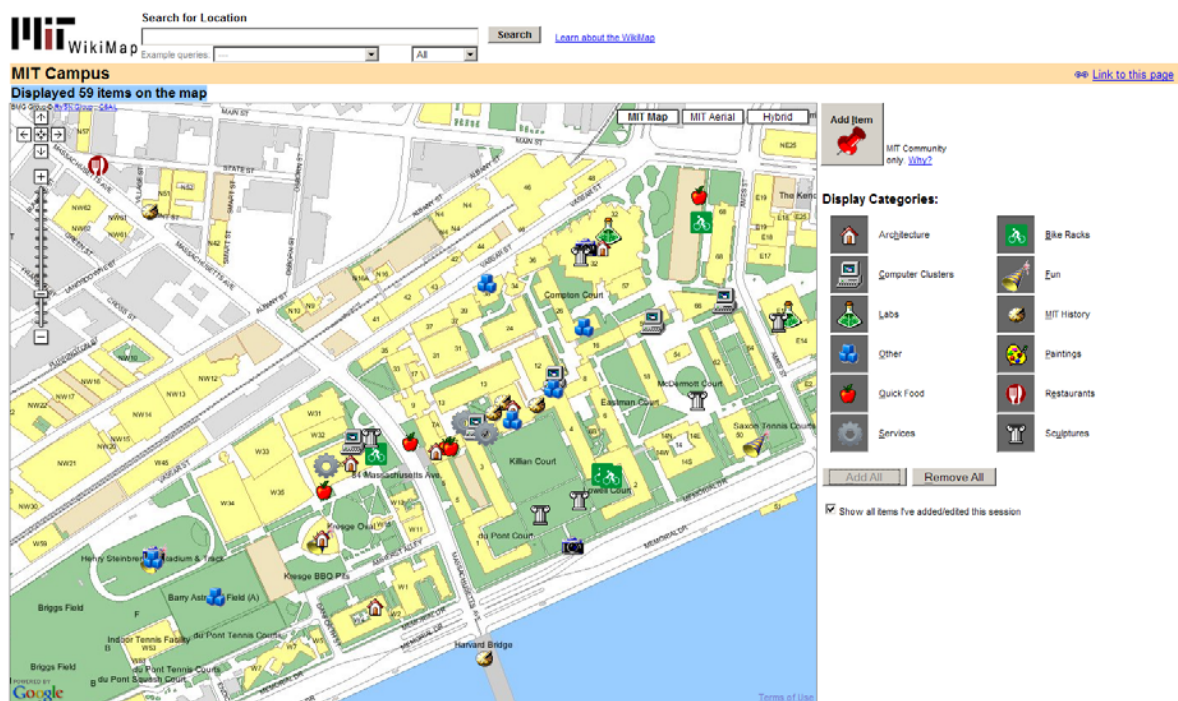


Figure 4-11: MIT WikiMap tool – visualizing 59 items of different category types on the map.

The MIT WikiMap was also built such that it would be a clean layer on top of MIT Maps. It is constructed by calling functions provided by the MIT Maps API. The abstraction layer between the two makes it much easier to extend and change in the future.

Finally, a goal of the MIT WikiMap was to make it easy to hook any data set into it and visualize information. This needed to be done in a modular way, and will be further explained in section 4.9.

4.6 Property Tool

In order to speed up and automate the interaction between the MIT Property Office and the people responsible for unaccounted-for items (see Figure 2-5), we added a layer in between the two. Our approach to this problem was primarily concerned with usability – in the end it was meant to help the Property Office become more efficient with their asset-tracking process.

There are two main pieces of the Property Tool. There is an administrative component that contains the utilities used by the Property Office. This includes their SumProp database, as well as the report generator and the people data in MITDB. The report generator accesses the people data in order to obtain email addresses, and sends an email with a nonce that grants the end-user temporary access to edit the locations of his/her items. This part was designed to allow for administrators to send asset information to responsible persons in a variety of ways. The administrator could choose to show or hide any number of fields, and could enable editing of the location, comment, and responsible person fields (the other fields cannot be edited by end-users in this system). Finally, it is possible to ask for the user to confirm the tag number for each of his/her items.

The second part of the Property Tool is for the responsible people. Once they receive an email from the Property office, they go to the interactive map by clicking on the link given in the email. The tool loads their items into the map, and the person can then change the locations of the items by either dragging icons or by typing the locations. Once this is completed, the updates are sent to the Property office. Part of the approach to the end-user side, therefore, was to assure that the original data would remain untouched. Only a data administrator can change the asset data by confirming the end-user's changes.

4.7 Other Applications

In the visualization builder we describe in chapter 7, the approach was to simply and cleanly hook into the data sets we federated using the MIT WikiMap as a visualization tool. Once the

marker bubble was generalized, it merely became a matter of searching for information in each of the data sets, and returning XML that could be interpreted and used to fill the bubble.

4.8 Back-End Design

The map tools all interact with the back-end using URLs and XML. Requests are made using a URL with variables encoded in the URL parameters. When the back-end has interpreted the request and performed the requested operation, XML is returned to the front-end. This XML is then parsed by the front-end and an appropriate action is taken. Figure 4-12 shows a simple diagram demonstrating this.

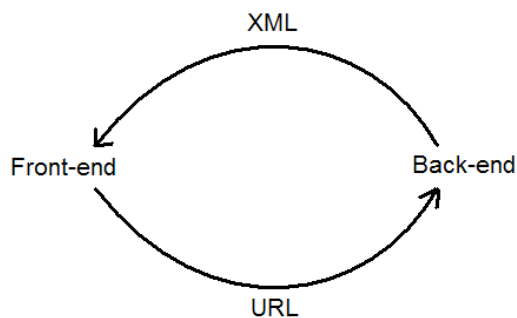


Figure 4-12: Front-end to back-end interaction.

All the data for our infrastructure is contained within a MySQL database called MITDB. It is accessed via PHP scripts that run on our server and are accessed via URLs. These scripts will take the URL parameters provided to it, and will use them to go down the appropriate branch of execution. Functions will be called to suit the needs of the query, and these functions will request information from the appropriate table(s).

This design is appropriate for three reasons:

- 1) XML is structured such that it is easy to interpret, yet flexible enough to allow us to include any information we chose, and to add fields as necessary.
- 2) URLs are also simple and are similar to XML in that they are structured yet flexible; one downside – we are currently restricted to GET requests, though we may add POST requests in the future.
- 3) PHP and MySQL are commonly used together and it is simple to set up a connection to a MySQL database using PHP. It is also easy to generalize this database connection such that any type of database can be swapped in.

4.8.1 Table Descriptions

We will now describe each of the tables contained in MITDB. It is worth noting here that though these are the tables that currently exist, it is simple to add new tables and information as we federate new data sets. Therefore, this is likely to be extended in the near future.

Tables can be broken down into a number of categories:

- 1) Data mapping table
- 2) Spatial data tables
- 3) Wiki tables
- 4) Property Tool tables
- 5) Equipment table(s)
- 6) Schedules table(s)
- 7) People table(s)
- 8) Hazardous materials table(s)
- 9) Visualization builder table
- 10) Category Icon table

Each of these categories will be described, and the tables and fields explained. Some of the tables require less detailed analysis than others, as we have already provided much of the needed information in section 4.3.

Data mapping table:

As described in section 4.3, we have a “datamapping” table that provides a way to store the pertinent information necessary to understand each federated data set. There are five columns – objtype, tablename, uniqueid, locationfield, and locationformat.

“objtype” is the object type we are federating (e.g. assets, equipment, schedules, etc.). The tablename is simply the name of the table for a particular data set (e.g. “class_sched” for Schedules data). “uniqueid” is a field (or multiple fields) representing the unique identifier within the data set. Therefore, it might be “TagNum” for assets data, and “Subject + Section” for schedules data. “locationfield” specifies the field(s) representing the location for each row in the table. For assets data, it would be “Location”, and for equipment data would be “Location + ‘-’ + Room”. Finally, “locationformat” is one of the four formats we specified in section 2.1.2 and 4.1 – “MIT”, “GPS”, “stateplane”, and “streetaddress” are valid entries in this column.

Spatial data tables:

gpoints
keywordspatial
rooms
spatial
subspatial

Figure 4-13: Spatial data tables.

gpoints: This table serves the purpose of storing GPoints for known space names on the MIT Campus. This is useful because if we have a data set where MIT room names or colloquial names are provided, we need a way to resolve these names to Massachusetts state plane coordinates. Therefore, we have a script (`getLocationGPointAll.php` in the WebScrape folder of `wikimap.csail`) that runs through the data model and looks at each space’s centroid coordinates, storing it to this table. This is done for a couple of reasons. First, it allows us to speed up the process of finding a location’s GPoint. We could look up each location’s GPoint from the data model as we need it, but this would be slow for situations where we need to return many locations. Secondly, it then allows us to not repeatedly store the GPoints for locations in multiple tables. Rather than needing to store the GPoint for each entry in the equipment and schedule table, for example, we can store it just in the one place and do a join to get all the information we need. The `gpoints` table contains three columns – `location`, `gpointx`, and `gpointy`. The `location` can be any formal space name on the MIT Campus, and `gpointx/gpointy` are decimal numbers to 6 places.

keywordspatial: This table provides a way to link keywords to locations and category names. It contains two columns – `name` and `keyword`. Therefore, if we wanted to link the keyword “coffee” to “Forbes family café”, we could add this entry. Then when we search for coffee, “Forbes family café” will appear.

rooms: This table contains all the formal space names on the MIT campus. It contains 9 columns – `name`, `building`, `floor`, `room`, `spacetype`, `minx`, `miny`, `maxx`, and `maxy`. Most of these column names are self-explanatory. The last four columns were briefly referenced in section 3.6.3, regarding point queries. They are used to quickly determine which spaces contain a particular coordinate on the map.

spatial: This table contains information about colloquial bindings to locations, and information about which category they can be classified into. There are three columns – `tag`, `iden`, and `cat`. “`tag`” is the colloquial name, such as “Badger Building”. “`iden`” is the formal name for the tag – in this case “E70”. “`cat`” is the category for the tag, which in this case is “Buildings”.

subspatial: This table contains information about subcategories of categories. It contains two columns – subclass and superclass. Giving an example is most helpful for this: if we wanted to include the category “East Campus” as a subcategory of “Residence Halls”, we would store that here. When we drill down into the “Residence Halls” category, we would see this subcategory. This allows us to easily add structure to our colloquial information.

Wiki tables:

wikiconfirm
wikiobj

Figure 4-14: Wiki tables.

wikiconfirm: This table contains information regarding whether objects added/edited in the MIT WikiMap have been confirmed. There are four columns – email, nonce, overallnonce, and timestamp. The email is the address the WikiMap requests from users upon adding or editing an item. The nonce is the unique identifier for the object being operated on. “overallnonce” is a unique identifier representing the user’s overall session. This is used in order to group items a user has added or edited in the same session. The WikiMap uses PHP session variables, and upon loading the map will assign the user a unique identifier. If the user edits an object, an email will be sent to the address provided. Any items edited in the session will have the same overallnonce, so if the user clicks the link in the email all these entries will be removed from the table. Furthermore, a new row will be added that contains the overallnonce and the word “confirmed” in the email column. If the user proceeds to edit another 9 objects, the overallnonce will be the same as the first object, and since that overallnonce is already confirmed, they will automatically be confirmed without sending additional emails to the user.

wikiobj: This table contains all the Wiki objects for the MIT WikiMap. The table fields are shown in Figure 4-15.

nonce	image
name	description
location	email
gpointx	timestamp
gpointy	confirm
category	active

Figure 4-15: Wiki object fields.

As referenced in Figure 4-1, Wiki objects obey the general structure proposed. The unique identifier is the nonce; location, gpointx, and gpointy have also been explained. “name”,

“category”, “image”, and “description” are self-explanatory (though “image” contains a path, not the image data itself). “email” contains the address for the person responsible for a change to an object. “timestamp” is simply when the change occurred. “confirm” is related to the wikiconfirm table, and is “1” for objects that have been confirmed, “0” otherwise.

“active” corresponds to the row in the table that is currently active for a particular nonce. If a single object has been edited 10 times, there will be 11 entries in the table (one extra for the initial addition of the object). Only one of these will be active, and this is an invariant that is maintained for all Wiki objects. This allows us to track all changes to an object, and potentially roll back to a previous version if necessary.

Property Tool tables:

altcontacts
assets
assets_temp

Figure 4-16: Property Tool tables.

altcontacts: This table allows us to bind contact information to particular assets. It contains five columns – TagNum, FirstName, MiddleName, LastName, and EmailAddress. If a particular asset has an alternate contact (different from the PI or RespPerson – see section 4.3.1), it is stored here, and loaded when an administrator looks at that asset.

assets: This table contains all the fields shown in Figure 4-2. It is updated via two methods: 1) when administrators upload new PRN files (containing asset information) to our system, and 2) when end-users make a change and an administrator confirms it.

Equipment / Schedules / People / Hazardous Materials tables:

Equipment	equipment
Schedules	class_sched
People	people
Hazardous Materials	hazmat

Figure 4-17: Table names for Equipment, Schedules, People, and Hazardous Materials.

These tables require less explanation because they were addressed in section 4.3. Similar to the assets table above, the fields in these tables are shown in Figures 4-3, 4-4, 4-6, 4-7, and 4-9. All of them have a location, but no gpointx and gpointy – this is provided by the gpoints table.

Visualization builder table:

marker_display: This table allows for the streamlined construction of marker bubbles in the MIT WikiMap and other applications. The fields are shown in Figure 4-18. How this data is used will be explained more fully in section 7.1.

ID	height
text	row
variable	col
type	objtype
colspan	grouping
align	

Figure 4-18: marker_display fields.

Category Icon table:

wikicategories: This table is used to store information about icons for the WikiMap, and it has 8 columns – name, path, width, height, anchorX, anchorY, windowX, and windowY. It will be explained more fully in section 7.2. To give a brief overview, it allows users to link icons to category names, such that interesting new icons can be displayed in the WikiMap.

4.8.2 History

As shown in the wikiobj table description, it is possible in this system to maintain a history of objects. Currently Wiki objects are the only objects for which we maintain a history, but it would be trivial to extend this to the other data sets. We simply need to have two additional columns – “active” and “timestamp”. As with Wiki objects, we would need to maintain the invariant that only one instance of an object is active at a time. The timestamp would allow us to distinguish between copies of each object.

4.9 Searching for Objects

Federating data sets would not be especially useful unless we had a good way to access the data we have federated. In this section we present the scheme used to hook into each data set, as well as how to resolve searches that are not constrained to one particular data set.

4.9.1 Constrained to One Data Set

We can search the federated data in a number of ways, and this list is constantly expanding as we determine useful searches for users. A list of the types of searches are listed below:

- 1) Building
- 2) Floor
- 3) Room
- 4) Colloquial space name
- 5) Routes from space to space
- 6) Wiki object – name
- 7) Wiki object – category
- 8) Equipment object – category
- 9) Equipment object – location
- 10) Asset object – name
- 11) Asset object – location
- 12) Schedule object – class name
- 13) Schedule object – open classes
- 14) People object – name
- 15) User that added an item

With such a large number of search types, it is necessary to create a structure whereby each branch can be distinguished from the others. In order to do this, we use keywords. This provides the flexibility for a user to type nearly using full sentences, and we can interpret it. Therefore, if the keywords “wiki”, “wikis”, “wikiobj”, or “wikiobjs” are typed in the search box, we know to restrict our search to just Wiki objects. Similarly “space”, “spaces”, “spacename”, and “spacenames” restrict to the building, floor, room, and colloquial space name set. For routes, “route” is the keyword we look for.

Once the top-level keyword narrows the set of possibilities, further keywords help us determine which branch to travel. Therefore, if we have typed “equipment location 10-250”, we go down the equipment branch, then see the keyword “location”, so we check for all matches that have location “10-250”. The benefit of using this method is that any combination of words will produce the same result. Therefore, “location equipment 10-250”, “10-250 location equipment”, or “10-250 equipment location” will all yield the same result.

Figure 4-19 displays the keywords at the primary and secondary level.

Wiki objects	wiki, wikis, wikiobj, wikiobjs	name, category
Spaces	space, spaces, spacename, spacenames	building, floor, room
Routes	route	to, debug
Equipment objects	equip, equipobj	location, description, category
Schedule objects	sched, schedobj	name, open
People objects	people, peopleobj	name, email
Asset objects	asset, assetobj	name, location
User	user	Name

Figure 4-19: Primary and secondary level keywords.

There are an arbitrary number of keyword levels – for any branch, it can continue expanding. Thus, we can type “sched open from 11 to 12 on Tuesday in 1”, and we would see the keywords “sched”, “open”, “from”, “on”, and “in”. In the case of “from”, “on”, and “in”, we use the words immediately to the right of the keyword to determine the parameters we need.

We also do not need to use every branch necessarily – “open from 11 to 12 on Tuesday in 1” would work without the “sched” at the front. This is because (as will be shown in section 4.9.2), Schedule objects will eventually be searched and the “open” branch will be found.

It is also worth noting once again the flexibility of this search. Therefore, valid alternatives to “open from 11 to 12 on Tuesday in 1” would be:

- 1) open classes from 11 to 12 on Tuesday in 1
- 2) open on Tuesday in 1 between 11 and 12
- 3) on Tuesday open classes in 1 between 11 and 12
- 4) on T open from 11 to 12 in 1
- 5) on T open from 11 to 12 in 1 blah blah blah (this works because the “in” keyword only looks one to the right, and therefore the “blah blah blah” is thrown out)
- 6) Many other combinations

Some of these example queries are demonstrated in the WikiMap. When a user clicks one of these, it runs that search without the user needing to click the search button.

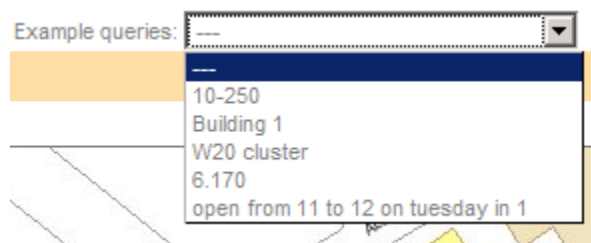


Figure 4-20: Example queries as shown on the MIT WikiMap.

4.9.2 Across Multiple Data Sets

It is also useful to have an easy way to hook into all of the data sets at once and search for items. Doing this is fairly simple. We simply need a function that imposes an order of operations. If no matches are found in the first data set, we move on to the next. If by the last data set there are still no matches, we return that. The order is as follows for the MIT WikiMap:

- 1) Wiki objects
- 2) Space names (colloquial and formal)
- 3) Schedule objects
- 4) Equipment objects
- 5) People objects

The other data sets either need to be accessed directly through one of the hooks shown in section 4.9.1, or are not accessible at all through the MIT WikiMap (hazardous materials is the only example of this due to security concerns).

4.9.3 Drop-down

It is also useful to have an intuitive way to view search matches. The MIT WikiMap and Property Tool have an incremental drop-down search box. This drop-down returns all current matches for the letters a user has typed thus far. This returns quickly enough for it to be useful to users.

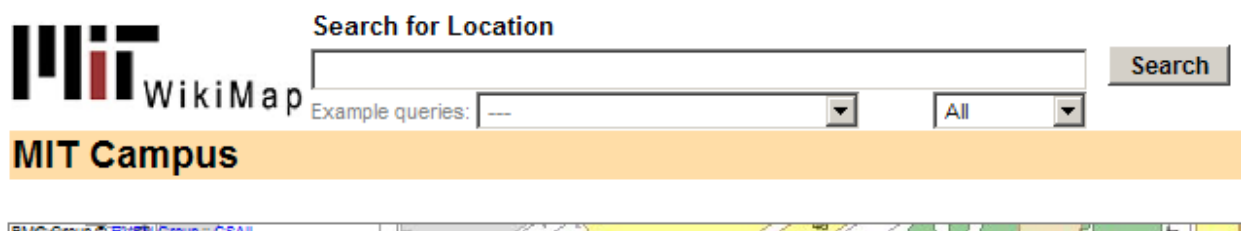


Figure 4-21: Search box.

When “All” is selected underneath the search box, Wiki objects and locations will appear in the drop-down. The reason that not all data sets are actually included here is because it would become too slow to react to a quick typist. Figure 4-21 shows the result when a user types “w”.

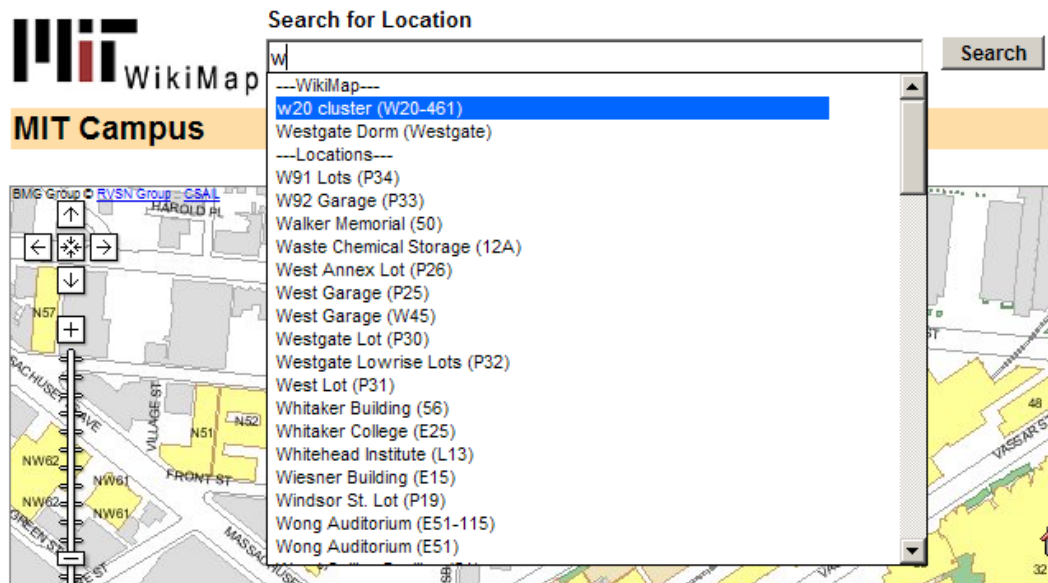


Figure 4-22: Returning matches for “w”.

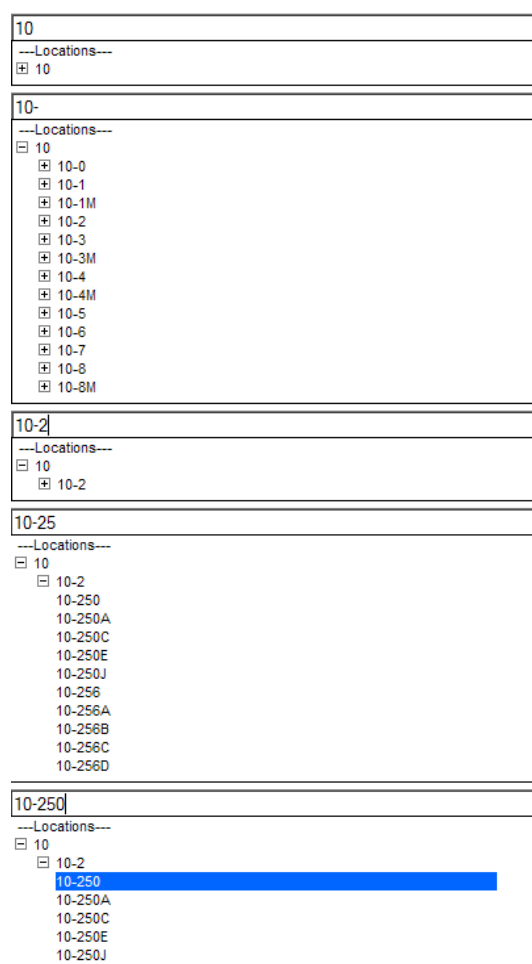


Figure 4-23: Drop-down sequence for typing “10-250”.

The user can then highlight the desired choice and click it. This drop-down is hierarchical as well. Figure 4-23 shows the drop-down’s display as the user is typing “10-250”. When the user has typed “10”, just building 10 is a match. Once the “-” is typed, the floors for building 10 will appear. Once “2” is typed, the only match is floor 10-2. After typing “5”, the hierarchical list further expands to show the space names that match. This is helpful to users for three main reasons. First, it allows the user to see the most important information. A drop-down without hierarchy might just show all the matches for 10, but this list would be large and the user would not be able to easily find what he/she is looking for. Second, it gives the user a better sense of the overall campus because of the organization. Finally, it is simply faster – if we are not displaying thousands of matches at a time, we can provide a better user experience.

It is also possible to use the hierarchy to navigate the structure. Figure 4-24 shows the display when we click the “+” next to 10-2. Naturally, this displays all spaces on the 2nd floor of building 10.

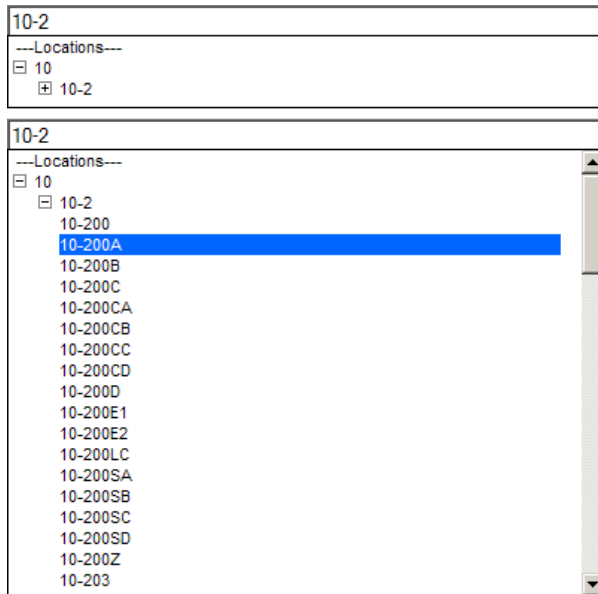


Figure 4-24: Clicking the “+” to expand the structure.

If we have typed “10-25” and click the “-” to minimize floor 10-2, then expand it again, only the spaces that match “10-25” are returned. This is the behavior we desire. Alternately, we could return all spaces on floor 10-2, but this would be confusing.

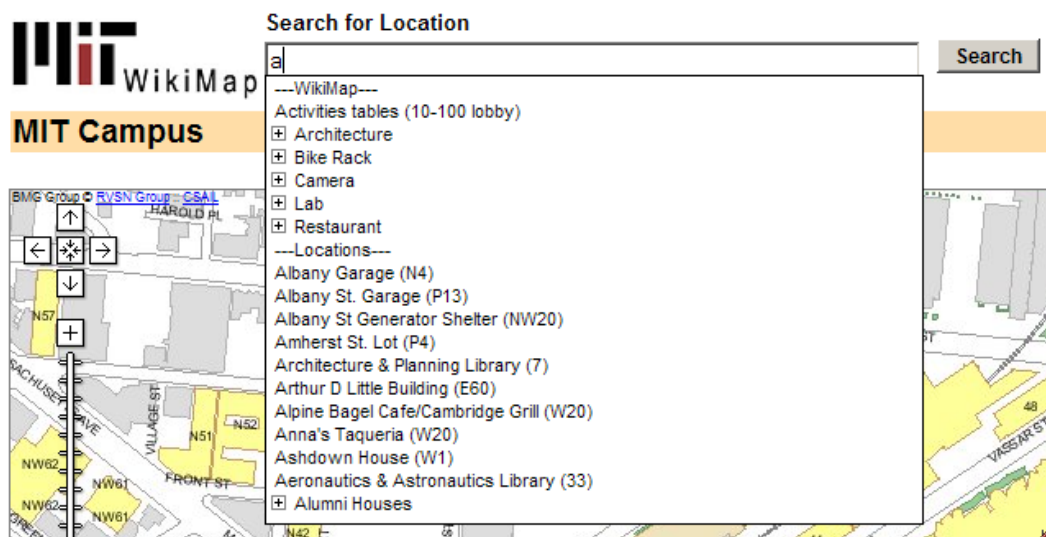


Figure 4-25: Typing “a” – we see a number of categories returned here.

As Figure 4-25 shows, this hierarchy is also useful for categories. As explained in section 4.8.1, the tables allow us to create hierarchy, and using the drop-down we have created, it is possible to visualize this easily.

Finally, changing the type selector below the search box changes what displays in the drop-down. Figures 4-26 and 4-27 demonstrate this. Changing to “Schedules”, we see class names appear when we type “6.1”.

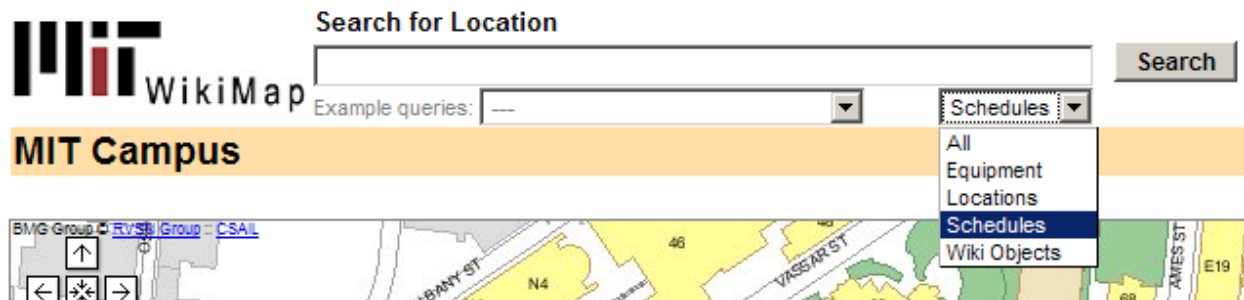


Figure 4-26: Changing the type of search to “Schedules”.

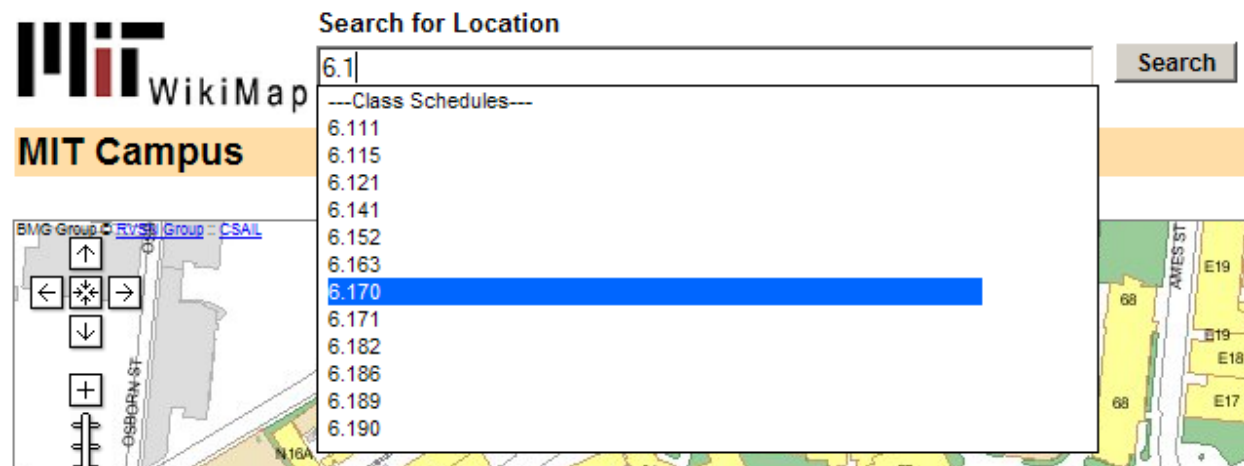


Figure 4-27: Typing “6.1” after having selected Schedules as the type.

Chapter 5

MIT WikiMap

This chapter explains in detail the implementation of the MIT WikiMap. It focuses on the user interface design aspects. We also discuss our user testing of the system, and the conclusions drawn from those tests.

Note: Sections of Chapter 5 are taken from “MIT WikiMap 6.831 GR6: User Testing and Final Report”.

5.1 Problem

We seek to address the problem that common spaces (such as a college campus) are host to a wealth of interesting and practical locations that are too often neglected or underutilized simply due to lack of awareness. As a collective, the denizens of a space encapsulate this information, but there is currently no scalable way of disseminating this knowledge to the individual (at least in an MIT context).

The MIT WikiMap is a solution to this problem. The WikiMap is an interactive, web-based application that allows collaborative, distributed tagging of geographical points on the MIT campus. (We restrict the WikiMap to MIT only for proof of concept; there is no reason why the WikiMap could not be applied more broadly). The WikiMap will serve as a collective record of the public’s knowledge of campus: any user may add an item to the map, at which point the item will be world-viewable. Any user may update or edit previously added items at any time, so an effectively used WikiMap will be perpetually correct and up to date.

The users of this application fall into two main groups: MIT community members and MIT campus visitors. MIT community members have varying levels of familiarity with the campus, and will be the primary contributors to the WikiMap, actively adding entries about locations with which they are familiar and updating those of others. Further, we expect that community members will also reap the most benefit, given the amount of time they spend on or around campus.

MIT campus visitors will generally be viewers of, rather than contributors to, the WikiMap. Our system offers visitors some degree of “self-guided-tourism” – the map will offer a wealth of

information regarding the unique cultural and artistic facets of MIT and where to find them. Also, the WikiMap offers pragmatic uses, for instance pointing out cafés or places to board public transportation.

The WikiMap offers 3 basic actions to the user:

- 1) Add an item to the map
- 2) Edit an existing item
- 3) View an existing item

Users' tasks largely consist of these operations. Important tasks include:

- *browsing the map* – a user may simply want to explore the campus by casually perusing the WikiMap. He may explore by location (e.g. West Campus) or by category (e.g. Paintings).
- *sharing a point of interest* – any time a user encounters a novelty or practicality of campus not already present on the WikiMap, she may access the WikiMap and add it.
- *choosing destinations based on proximity* – users may find locations which offer multiple amenities, e.g. food near a lecture hall, or a computer cluster near bike racks.
- *landmark based navigation* – as the WikiMap becomes populated with distinct visual entities from around campus, a user (or eventually a computer algorithm) may give directions or plan paths based upon particular landmarks.

5.2 Overall Design

The MIT WikiMap must conform to all of the properties outlined in section 4.1. First and foremost, however, it must be intuitive and flexible. It must have an inviting user interface that begs to be used and experimented with. Since the success of this tool depends on how much the MIT community uses it and adds to the corpus of data, the user interface is a critical element.

Figures 5-1 and 5-2 demonstrate the interface for the MIT WikiMap.

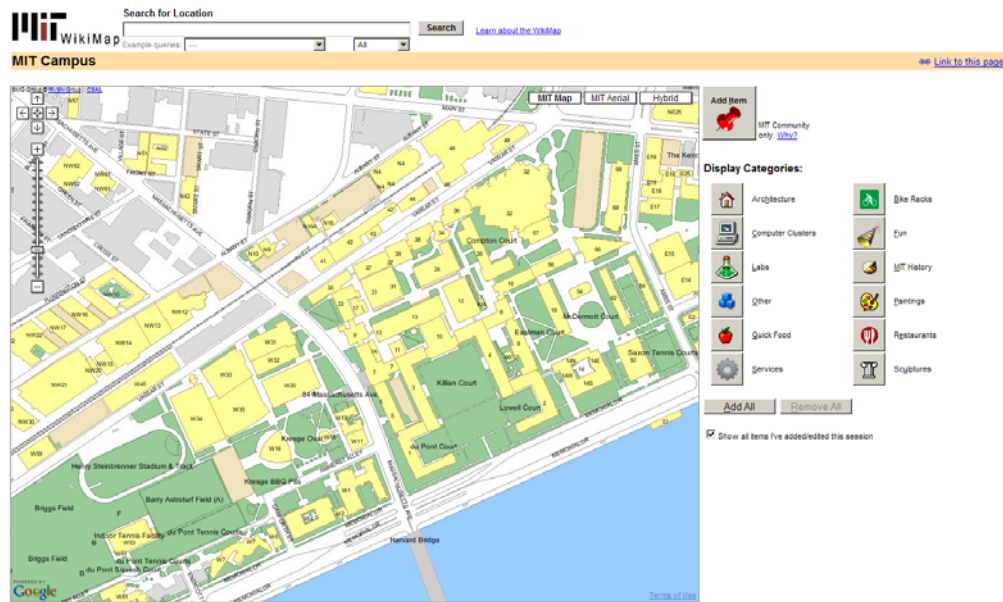


Figure 5-1: Refreshed WikiMap interface.

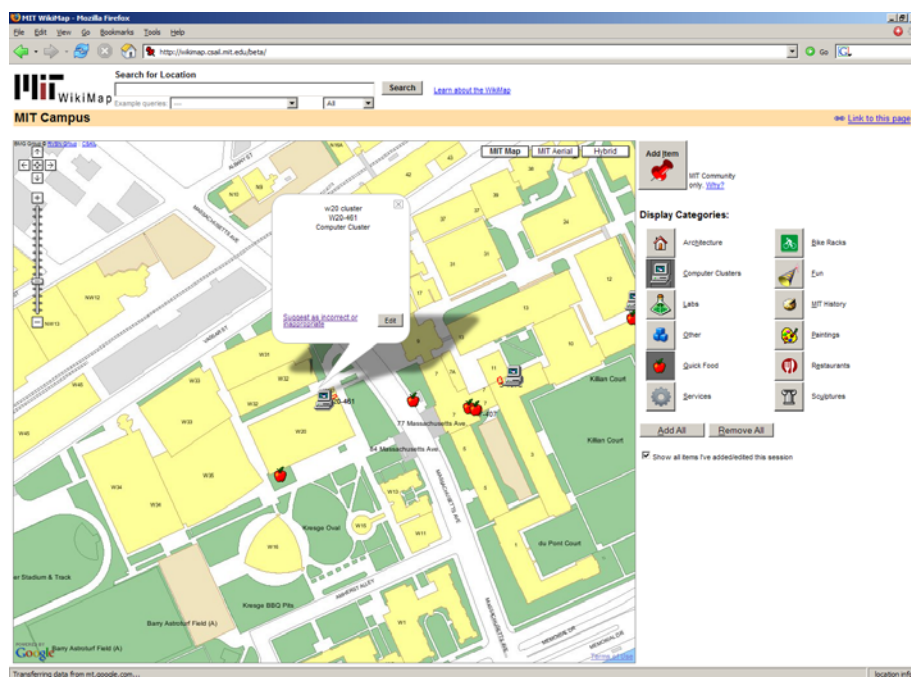


Figure 5-2: WikiMap while viewing items.

5.2.1 Front-End / Back-End Interaction

The front-end and back-end of the MIT WikiMap interact in a structured and modular way. All communication between the two occurs with URLs and XML files. When the front-end would like to display something, a URL is accessed through a JavaScript asynchronous fetch. This will

access a script on the back-end that queries the MITDB database and returns the information in XML format. This XML is then parsed by a callback function on the front-end and the map changes.

5.2.2 Implementation

The MIT WikiMap is built using Javascript, PHP, and MySQL. Implementing it in this way is lightweight and therefore it can easily be loaded into a browser. The tradeoff is that certain functionality is restricted since Javascript's capabilities are not as wide as Java.

One example where our implementation choice restricted usability is that the map does not scroll when an icon is dragged outside the visible region of the map. Though possible, it would be extremely complicated and break the abstraction of the Google/MIT Maps API. Scrollable maps are much easier to implement in a Java applet. We decided against that option for two main reasons: 1) It would add a large overhead on the initial load of the system. We consider this a major usability problem, and 2) There are security issues with Java applets that are difficult or even impossible to manage in all situations.

The images displayed by the MIT WikiMap are generated by IMS and cached on our server, as described in section 3.6.2.

The MIT WikiMap has a MySQL database for its back-end. Data is stored persistently in the database as users add and edit items. The data for the objects is stored in a table called "wikiobj". There is another table called "wikiconfirm" that keeps track of which objects are awaiting confirmation. Finally, there is a table called "wikicategories", which binds category names to icon images. This organization makes it easy to add and edit categories.

The implementation also relies on session variables for storing user information. When a user accesses the web page, a session variable called "overallnonce" is generated. This is a unique nonce to identify that user for the session. Even if the user refreshes the page, or goes back or forward with the browser, this session variable will be there. Only if the browser window is closed will this session expire. We made this design decision in the implementation in order to increase user control and freedom.

Once a user adds or edits an item for the first time, he/she must enter an email address. Since we are using a session variable, this only needs to be completed once. A confirmation email is then generated and sent to the address provided. Additionally, the entry is added to the database,

although the “confirm” column of the new entry is set to 0 (off). Only once the user confirms the object will the entry’s “confirm” column go to 1 (on).

Email confirmation is flexible. The user may click on the confirmation at any time. For example, a user may add/edit 10 different items then click on the link via email to confirm. Since all of the objects were changed in the same session, they will all be confirmed at once. Alternately, a user may add/edit an item, then click on the link provided in the email, and continue editing items. Future adds/edits in this session will be automatically confirmed, since the user already confirmed his/her identity.

We decided that we wanted the system to be as real-time as possible. We took advantage of many of the asynchronous capabilities of JavaScript. When a user clicks a category to load it onto the map, an asynchronous call is made to our database, and XML is returned. This XML is then parsed and displayed to the user appropriately. This system design means that the user never needs to reload the page in order to view changes; they are all available immediately.

Furthermore, when a user clicks an icon to load its information, it will first display a local version of the information (stored client-side in JavaScript variables), and then will query the database to find the most up to date information for that item. If another user changes the information for a particular icon, that change will be immediately visible to other users (after clicking on that icon) - no page reloads are necessary. Note this only occurs if a user is viewing, rather than editing an item, since we would not want to overwrite a user’s edit.

This real-time nature leads to a slight usability issue – a flicker upon opening an item’s information. This is caused by the call to the server asynchronously coming back and refreshing the information displayed to the user.

When an item’s location is changed (after the user drags and drops it), the location matches for that point are displayed. This information is retrieved by doing a point query (explained in section 3.6.3), returning matches for the provided coordinates. There are four layers that we use in order to return these locations - rooms, greens, parking, and landmarks. The rooms layer will return all rooms that contain that X,Y coordinate. As a consequence, multiple rooms will be returned if the coordinate goes through multiple floors. The greens layer allows points such as Killian Court or McDermott Court to be identified. Parking is for parking lots and garages, and landmarks are for commonly known locations, such as 77 Mass Ave or the Harvard Bridge.

The one complication here is that IMS returns space codes (for greens, parking, and landmarks), rather than colloquial names. For example, IMS calls Killian Court “G6”. In order to resolve this, we have another table (“spatial”) that stores bindings between space codes and colloquial

names. After doing a point query to the various layers, we send off another asynchronous call to a script that returns these bindings as XML. We then replace the space codes with the colloquial names if they are available. Similarly, we make sense of the space codes when we know a more human-readable form for the location. We did this for lobbies, corridors, stairs, and elevators. Instead of returning “10-100LA” to the user, we return “10-100 lobby”.

5.3 User Interface Design

Though the user interface should be intuitive to users, we also include help and documentation for the MIT WikiMap. For comprehensive information and instructions for use of the WikiMap, users can click the “Learn about the WikiMap” link next to the Search box.

5.3.1 Map Browsing

Navigating the map is central to the user experience. Our implementation is modeled after Google Maps, which we deem to be the most usable web-based mapping tool. We provide direct manipulation controls that allow the user to drag the map to pan and double-click a point to re-center the map. A zoom bar is provided to change the zoom level. Users may also submit a location in the search field at the top of the page, and the map will zoom and pan appropriately to display the result.

When an object is moused over, the contour of the enclosing space is drawn around it. This provides the user a better sense as to the surrounding area of the object.

5.3.2 Adding New Objects

The Tack



For the Add Item task, we introduced an affordance for pinpointing a location. We chose a tack because it is a well-understood metaphor for representing information on traditional paper maps. It also provides a visual representation for new items that is distinct from existing category icons. Using direct manipulation, the tack is simply dragged across the map to designate the new item location. Tacks are created via a prominently placed button at the top corner of the map. Multiple features are implemented for user freedom and efficiency:

- 1) Every time the tack button is pressed, an additional tack is added to the map. This allows users to plan multiple items. For example, many tacks can be positioned at their intended locations before entering descriptive information for each one.
- 2) We included a “Delete Tack” button that removes the tack from the map and cancels an Add-item in progress.
- 3) New tacks are positioned in the center of the viewable region of the map. This increases efficiency when used in conjunction with the search box. A location search followed by an Add-item action, will position the tack directly on top of the desired room.

Design Alternatives:

The earliest design iterations included modes for adding and viewing items. This had poor learnability. The WikiMap is a new concept and users are not familiar with the operations available. Modes restrained the user from exploring all functionality in viewing, adding and editing items. The result was frequent mode errors, for instance, attempting to add items while in view mode and vice versa.

Location Bubble

We treat the campus as 2.5D (layers of 2D stacked on top of each other), e.g. by floors within a building. Consequently, choosing a location may sometimes be ambiguous. We chose to alleviate this issue with a location-selection bubble that provides a user with all the possibilities at a particular 2D point.

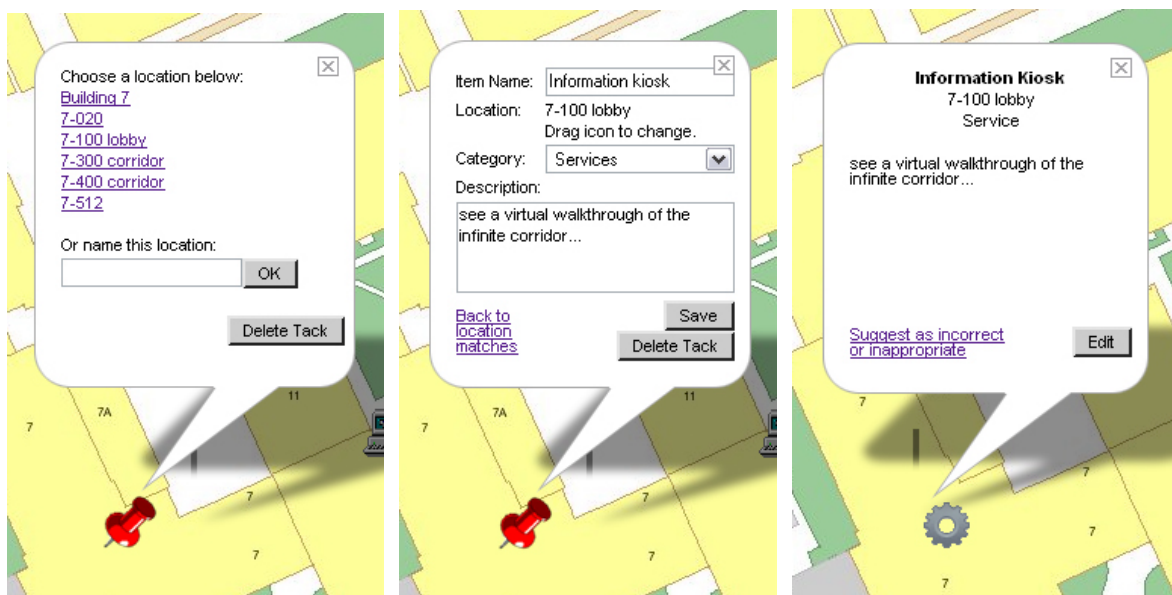


Figure 5-3: (left) location bubble; (center) editing item info; (right) saved Service item.

For instance, dropping a tack on 10-250 also offers 10-455 and the hallway below the lecture hall, among other possible rooms. The user may alternatively choose the broader “Building 10” if the entry pertains to the entire building.

In order to speak the user’s language, we incorporate room types into location choices (mentioned in section 5.2.2). For instance, instead of the formal space code “7-100LA,” the location bubble displays “7-100 lobby.” A text field is also available if the user prefers to enter a colloquial location name.

Design Alternatives:

If a user were to type a valid location name in the text field, we considered automatically moving the tack to that location. This would act as an alternative to dragging the tack across the map. In the end, we decided against this feature because it would be disorienting to users and would be confused with the ability to submit colloquial names.

5.3.3 Editing Objects

When an item is edited, the previous versions of the item in the database are left intact. The only part that changes is the “active” column of those entries. Only one entry in the database for a particular object (identified uniquely by a nonce) may be active at any time. This obeys the invariant laid out in section 4.8.2 History. This facilitates rolling back an entry if a user inappropriately edited an item.

Category List

For the sake of consistency we decided to restrict category edits to administrators, for example, to prevent redundant item groupings such as “buildings” and “architecture”. Future extensions may include hierarchal category lists to allow for greater variety of display selections and a form for user suggestions of categories.

Design Alternatives:

Allow users to input their own category if desired. This allows user freedom but would lead to a long, cluttered category list with inconsistent or absent icons.

Changing Location of an entry

In the spirit of the Wiki, entries may need to be moved to new locations, while still retaining the rest of their data. We address this ‘user freedom’ feature by allowing the entry icons to be

dragged. Upon a drop of an existing entry, we present the user with a location-selection bubble (as when adding an item for the first time). A special case arises when the user drags an icon while editing the entry. Here, we store the current state of the edit so that it is once again available upon drop. Edit state is also maintained if the user closes the edit bubble, or views other existing items before saving.

5.3.4 Searching For Objects

We incorporate a drop-down menu with incremental search of all possible locations and Wiki objects. This ensures users will select a valid search item before submitting the query. A successful search re-centers the map and either pops up the marker bubble for a Wiki object, or draws a contour of a selected space.

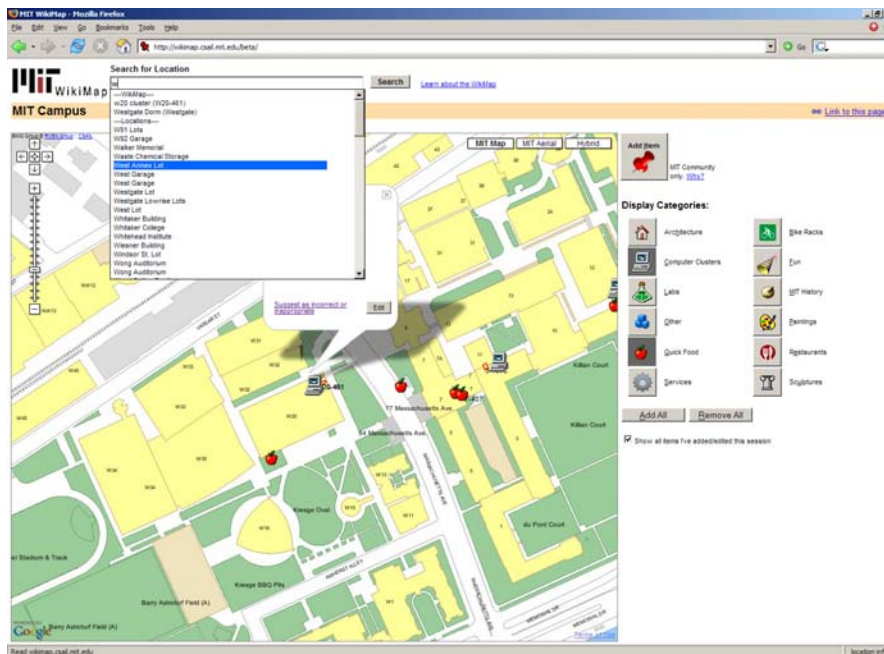


Figure 5-4: Drop-down triggered by typing “w” in the search box.

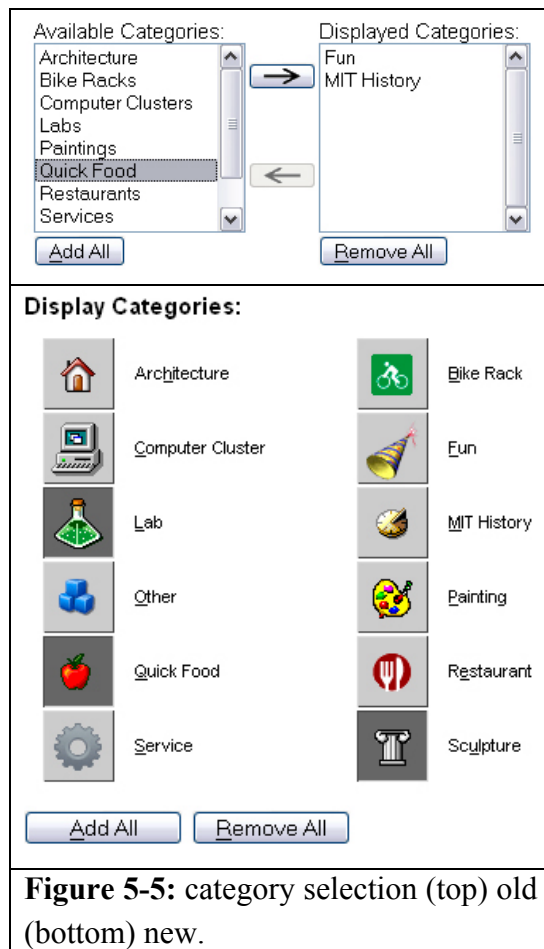
Design Alternatives:

In earlier iterations, the search box was intended for multiple query types. For example, a valid query would be “bldg 10 food bike racks” in an attempt to display all bike racks and places to eat in the area surrounding building 10. We decided implementation of such a sophisticated search tool was beyond the scope of this project. Complications would arise in recognizing which part of a query corresponds to location, and which part to category, and how to provide error recovery when search queries cannot be resolved.

5.3.5 WikiMap Categories

Toggle-button Category selection

We chose to implement category selection as a set of toggle buttons. Our user tests clearly showed this was desirable, and in designs that pre-dated this selection idiom, users would ask why we didn't have this.



The buttons offer a clear affordance for clicking and are better mouse pointer targets than, say, a hyperlink. Putting icon images on the buttons acts as a visual draw and makes clear the connection between buttons and map icons. Toggle buttons also offer easy undo: the user clicks in exactly the same place to reverse a display or hide action. To increase efficiency, we implemented “Add All” and “Remove All” options, along with keyboard shortcuts for every button.

Design Alternatives:

An alternative K of N selection model can be seen in our original selection UI, which uses two listboxes to display the available and displayed categories. This UI scales better with the number of categories, addressing a concern we had in early designs. However, this UI has many drawbacks. Adding a category requires at least a double click, and undoing an action would require the user to move the mouse. Further, the mapping between category names and icons was established by a small legend, which led users to erroneously click on the icons in the legend hoping to display items from that category.

5.3.6 Storing User Information

'Items I've added this session'

A consistency issue arose when prototyping the Add-item feature: What is displayed if the category of the added item is not selected as a Displayed Category? For example, the user adds a Sculpture item to the map, but only Quick Food and Bike Racks are selected.

In every round of user tests (since the paper prototype), we found that users frequently re-check items immediately after saving to ensure the information is displayed as expected. As such, we decided added item icons should always be displayed. This provides visual feedback for a successful save, and allows users to easily view and edit their contributions (user control).

To deal with consistency issues, we included the check box “Show all items I've added/edited this session” that decouples selection of display categories from the user's personal contributions.

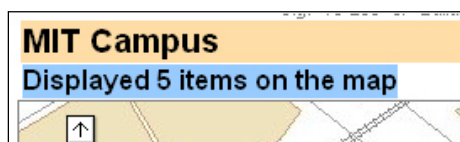
Design Alternatives:

One alternative was to enforce consistency by automatically selecting the category of the added item as a Displayed Category. We decided it would be disorienting for a user's category selections to be over-ridden. Instead, users are given complete control over category display.

5.3.7 System State

Data entered by the user is persistent. When a user clicks to edit an item, changes something, then clicks off the edit bubble, the state will be the same when the user returns to the item's information. This is done by maintaining two sets of the item's data - one that is editable and one that is not. Any time a change is detected (using `onKeyUp` and `onMouseUp` events), the editable version will change to reflect what the user did. If a user saves his/her changes, the editable version of the data will overwrite the non-editable version. Alternately, if the user cancels changes, the non-editable version will overwrite the editable version. This can be used even for location changes. Therefore, if a user drags an item to another location, then decides to put it back, there is a link he/she can click on that will jump the item back to its original location. We made this design decision in order to reduce errors and increase user control and freedom.

System Status



Some actions, such as displaying all sculpture items, may have no visible effect on the map, either because no such

items exist, or the items are outside the viewport extents. To ensure users receive immediate feedback, we decided to implement status messages displayed prominently above the map. A blue background hue (contrasting with map colors) and bold type add to visibility. This feature gives confirmation an action was completed successfully and speaks the user's language. Displaying a category will return the message "Displayed n items on the map." Pressing save on an edit will be confirmed with "Item successfully saved." Messages are removed after 10 seconds to avoid screen clutter or messages taken out of context.

5.3.8 Deterring Vandalism

The ability to add and edit items is restricted to members of the MIT community. This distinction is made for two reasons: 1) community members are assumed to have sufficient familiarity with the campus that they can appropriately add items, and 2) this allows the WikiMap to deter vandalism.

We take several measures to deter vandalism. First, we make a user accountable by requiring a valid MIT email address to be entered for a user's edits and additions to be finalized. This allows the WikiMap administrators to associate every change to the system with a member of the MIT community so that proper actions may be taken against inappropriate users.

Second, we put the power to delete an entry exclusively in the hands of the administrators. This prevents a malicious user from arbitrarily removing other peoples' work from the WikiMap. Any user may flag a WikiMap entry for deletion, but the action doesn't occur until an administrator approves it.

For efficiency reasons, users with add/edit permissions (MIT community members) will get only one email asking them to confirm all changes they made during a WikiMap session. This email is sent after completing the first change, but the confirmation will apply to all changes during the session, no matter when the user confirms. Only after this confirmation will a user's changes be publicly visible on the WikiMap.

5.4 User Testing

User testing was a major part of the MIT WikiMap's evolution. The user interface changed dramatically from our first user test to our last. This section outlines our user testing procedure and results.

5.4.1 Procedure

We conducted our user test by finding users, briefing them, and then letting them complete tasks at a computer. Since the WikiMap is web-based, we only needed to ensure that the computer had access to the Internet. We observed the users during their interaction and took notes on their behaviors.

Finding users from the MIT community was trivial. We asked several friends and acquaintances, both undergraduates and graduates, to be users for us. The bulk of our users were from this group, as is reflective of our anticipated demographic for WikiMap usage. To represent MIT visitors, we asked a college student who was visiting MIT for a day to be a user.

We started our user briefing with an overview: “The MIT WikiMap is a web-based, collaborative way of distributed tagging features of the MIT campus by location.” We then asked our users if they were familiar with the concept of a Wiki. Those who were not familiar were briefed on how a Wiki allows public editing of common resources. Next, we asked our users if they had used Google Maps, after which we modeled our map zooming and panning UI. All had used Google Maps.

We provided no demo, hoping our interface could be easily learned. Before performing tasks, most users desired a few moments to explore and experiment with the interface, which we granted them.

We asked each user to perform 3 tasks. MIT community members were asked to:

- 1) Find a place to eat with bike parking nearby.
- 2) Add an item to the map at a specified location (different items and locations for each user).
- 3) Edit an existing item on the map.

Non-community members currently do not have add/edit privileges in order to thwart anonymous vandalism, so their tasks differed:

- 1) Locate 26-100 exactly.
- 2) Find a place to eat with bike parking nearby.
- 3) Find something interesting to see (artwork, sculpture, etc. at the user’s discretion) on the eastern side of campus.

We then reminded the user that we were testing the interface, not the user.

5.4.2 Results

Cosmetic:

None found.

Minor:

A user complained some icons like “Fun” blend into the background. We could solve this by redesigning the icons (for example, having drop shadows).

Text asking a user to enter an email address to validate the session was not easily understood at first read. We solved this by rewording the dialog box to be more concise.

Major:

Many users were unconsciously reluctant to zoom the map. All users had used Google Maps before, so all users knew that the map was zoomable and how they could accomplish this. Despite this, new users tended to remain at the default zoom level, regardless of the number of items they displayed on the map or the area they were interested in. Reminding users that they may zoom the map whenever more than a certain number of icons become displayed could alleviate this. Alternatively, zooming could be made more accessible by having the scroll wheel control the zoom level.

Users were often quick to enter their own place names. When presented with a list of locations in the location bubble, many users went straight to the “Name this location” field. Sometimes, the user did this correctly (e.g. to put the item “DDR machine” in the “Game Room”), but other times users entered a room location simply because they did not see their room listed (possibly because the tack was not in the right place). The users may have expected the tack or item to relocate to the correct location, but we made the decision not to do this as it could be disorienting, especially if the entered location was off-screen. We tried to mitigate this mistaken usage by explicitly stating to the user that their best action may be to reposition the tack.

Catastrophic:

None found.

5.5 Reflection

Upon reflection, the iterative design process was critical to the success of our interface. We can remember how pleased we were with our initial design, and we now all look back at it aghast at how much more usable and elegant our current design is.

We chose to paper prototype virtually our entire front end. The benefit of this was that it helped us realize what parts of our back-end were unrealistic (e.g. a super-smart search box that would parse complex search expressions). The paper prototype session also had its drawbacks. For instance, until final user testing, we had a false sense of security that our status messages would guide the user. Indeed, they did in the paper prototype, but that probably owed to the fact that they were physically placed onto the interface, thus drawing attention to them. In our user testing, we found that users barely noticed the status messages, let alone took their advice.

Another key to the iterative design was to have frequent discussion. We spent hours discussing design decisions such as whether tacks should automatically relocate if the user renames a space to a known string or whether requesting an item deletion should be restricted to members of the MIT community. In the course of these discussions, we were able to flesh out ideas and bring to life others.

We also received good feedback from others along the way, notably Professor Seth Teller.

Chapter 6

Property Tool

This chapter discusses the Property Tool, an application developed for the Property Office using the infrastructure we have put in place.

Note: Some of this chapter is adapted from Location-Aware Asset-Tagging Using Crickets [11].

6.1 Problem

The goal of this project was to assist the MIT Property Office with the process of continuously tracking assets on campus. They do this by placing a barcode on any movable item purchased by an MIT department over \$1000, and maintain these “tagged” assets in a database.

The Property Office would often spend months tracking down unaccounted-for items, and the process of doing so was extremely tedious. Therefore, in order to help streamline this process, we built a tool that can manage unaccounted-for items in a database, find the appropriate email addresses corresponding to each item, and email the responsible persons in an intuitive manner. Once these emails are sent, the responsible persons can visualize their items on a map, and update the locations to reflect the most recent status.

6.2 Overall Design

There are two pieces to the design of the Property Tool – the administrative side, and the end-user side. Both parts run on the same website – however, the administrative side requires a username and password to access, whereas the end-user side requires a valid nonce (which is tied to a set of unaccounted-for items).

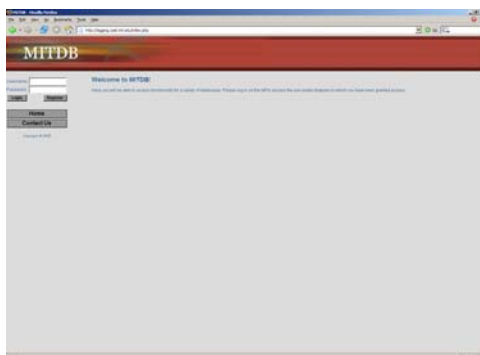


Figure 6-1: Main screen for <http://tagging.csail.mit.edu/index.php>.

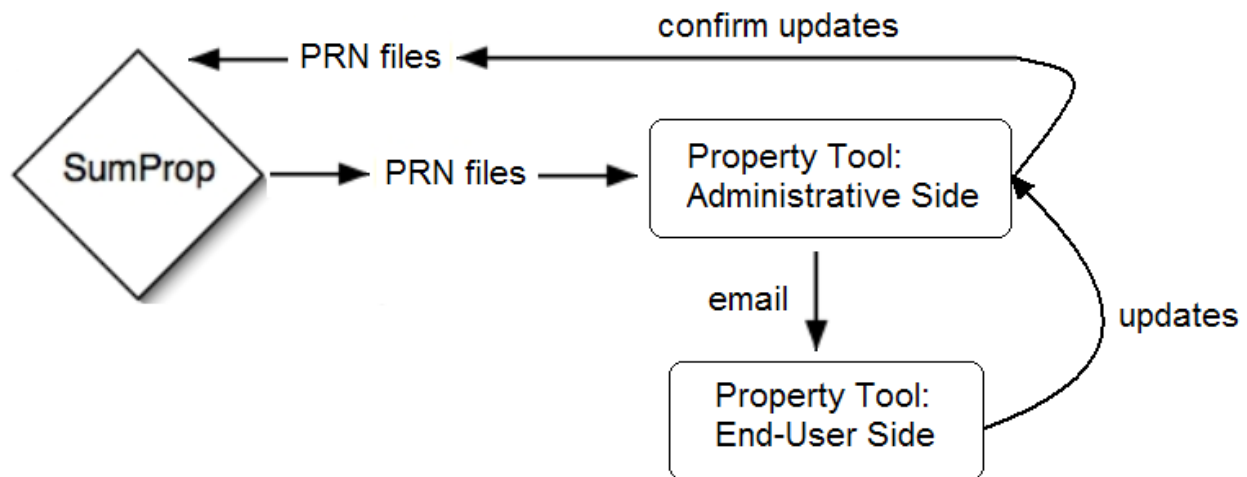
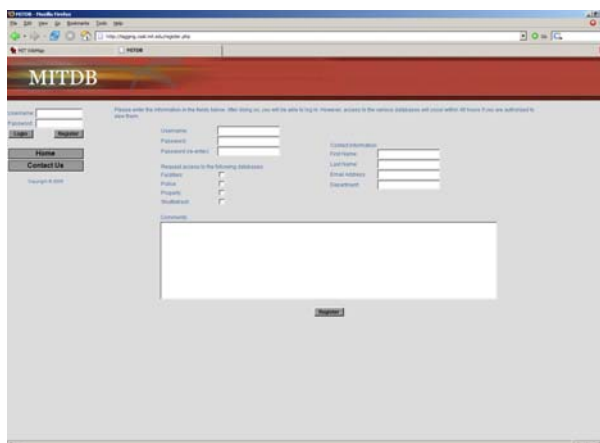


Figure 6-2: Property Tool overall process.

The process works as in Figure 6-2. SumProp is used to generate PRN files of unaccounted-for items. These PRN files are uploaded to the administrative side of the Property Tool, which then generates emails to responsible persons. Once the sent items are resolved, the Property Office must confirm these updates. Once they are confirmed, PRN files are generated that contain the updated information. These can then be uploaded to SumProp.

6.3 Administrative User Interface Design

This section will detail the functionality of the administrative side of the Property Tool. In designing this, we wanted it to both be flexible and user-friendly. Sometimes tradeoffs needed to be made between these two, but overall both goals were achieved.



To start, an administrator in the Property Office must sign into the tool. If a username and password has not already been set up, he/she may register for the site, as shown in Figure 6-3.

Figure 6-3: Registration page for MITDB.

6.3.1 Connecting To SumProp

Once signed in, the user must then upload a PRN file to the website in order to begin the process of following up on unaccounted-for assets. If a PRN file already exists, he can use that; otherwise he must connect to SumProp to generate a new file. Figure 6-4 shows the upload process.

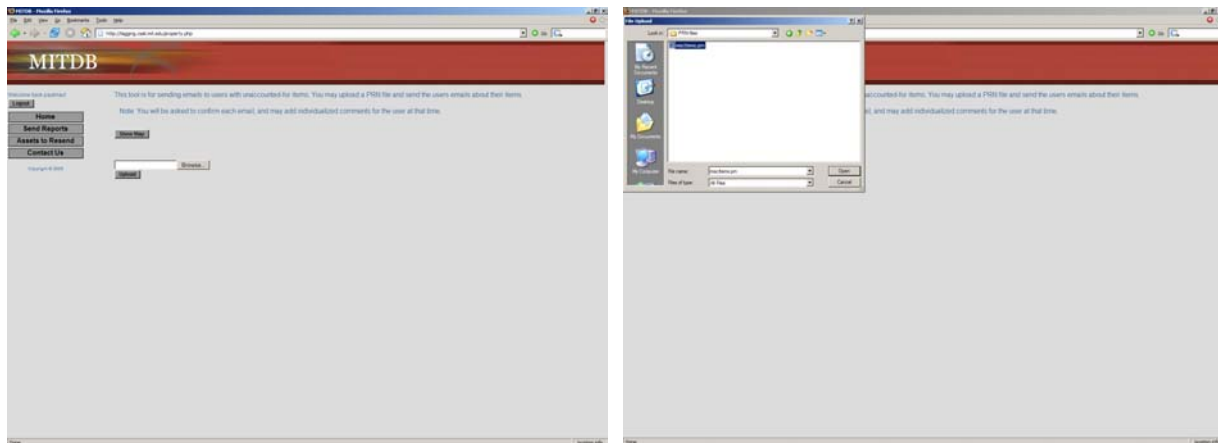


Figure 6-4: Uploading a PRN file to the website.

6.3.2 Visualizations

Once a PRN file is uploaded, `parseDelimited.php` parses it, adds it to the “assets” table in our database, and the output is displayed on the screen. This output can be displayed in table and map form. The default behavior for the administrative side is to hide the map, and simply display the table. The reason for this is that if an administrator is trying to quickly send a large group of assets, the map visualization is not desired. Figure 6-5 shows the site when the “Show Map” button is clicked.

The map uses the same infrastructure as the MIT WikiMap and other applications. An administrator can customize the marker bubble on the map. This will be demonstrated in section 7.1.

The administrator is able to change the categories viewed in the table using the Add/Remove box above it. The table can also be sorted by clicking on any of the table headers.

MITDB

Welcome back paulmad

Logout

Home

Send Reports

Assets to Resend

Contact Us

Copyright © 2008

This tool is for sending emails to users with unaccounted-for items. You may upload a PRN file and send the users emails about their items.

Note: You will be asked to confirm each email, and may add individualized comments for the user at that time.

Search for Location

Hide Map

Search

MIT Map MIT Aerial Hybrid

SCANNER,OPTICAL 1-110

Tag #: 0282648

Resp Person: PAOLUCCI,DORA M

Pl: NELSON,KEITH A

77 Massachusetts Ave

Lowell Court

4th Pond Court

McDermott Court

Eastman Court

Knickerbocker

W20

W19

W18

W17

W16

W15

W14

W13

W12

W11

W10

W9

W8

W7

W6

W5

W4

W3

W2

W1

W0

W-1

W-2

W-3

W-4

W-5

W-6

W-7

W-8

W-9

W-10

W-11

W-12

W-13

W-14

W-15

W-16

W-17

W-18

W-19

W-20

W-21

W-22

W-23

W-24

W-25

W-26

W-27

W-28

W-29

W-30

W-31

W-32

W-33

W-34

W-35

W-36

W-37

W-38

W-39

W-40

W-41

W-42

W-43

W-44

W-45

W-46

W-47

W-48

W-49

W-50

W-51

W-52

W-53

W-54

W-55

W-56

W-57

W-58

W-59

W-60

W-61

W-62

W-63

W-64

W-65

W-66

W-67

W-68

W-69

W-70

W-71

W-72

W-73

W-74

W-75

W-76

W-77

W-78

W-79

W-80

W-81

W-82

W-83

W-84

W-85

W-86

W-87

W-88

W-89

W-90

W-91

W-92

W-93

W-94

W-95

W-96

W-97

W-98

W-99

W-100

W-101

W-102

W-103

W-104

W-105

W-106

W-107

W-108

W-109

W-110

W-111

W-112

W-113

W-114

W-115

W-116

W-117

W-118

W-119

W-120

W-121

W-122

W-123

W-124

W-125

W-126

W-127

W-128

W-129

W-130

W-131

W-132

W-133

W-134

W-135

W-136

W-137

W-138

W-139

W-140

W-141

W-142

W-143

W-144

W-145

W-146

W-147

W-148

W-149

W-150

W-151

W-152

W-153

W-154

W-155

W-156

W-157

W-158

W-159

W-160

W-161

W-162

W-163

W-164

W-165

W-166

W-167

W-168

W-169

W-170

W-171

W-172

W-173

W-174

W-175

W-176

W-177

W-178

W-179

W-180

W-181

W-182

W-183

W-184

W-185

W-186

W-187

W-188

W-189

W-190

W-191

W-192

W-193

W-194

W-195

W-196

W-197

W-198

W-199

W-200

W-201

W-202

W-203

W-204

W-205

W-206

W-207

W-208

W-209

W-210

W-211

W-212

W-213

W-214

W-215

W-216

W-217

W-218

W-219

W-220

W-221

W-222

W-223

W-224

W-225

W-226

W-227

W-228

W-229

W-230

W-231

W-232

W-233

W-234

W-235

W-236

W-237

W-238

W-239

W-240

W-241

W-242

W-243

W-244

W-245

W-246

W-247

W-248

W-249

W-250

W-251

W-252

W-253

W-254

W-255

W-256

W-257

W-258

W-259

W-260

W-261

W-262

W-263

W-264

W-265

W-266

W-267

W-268

W-269

W-270

W-271

W-272

W-273

W-274

W-275

W-276

W-277

W-278

W-279

W-280

W-281

W-282

W-283

W-284

W-285

W-286

W-287

W-288

W-289

W-290

W-291

W-292

W-293

W-294

W-295

W-296

W-297

W-298

W-299

W-300

W-301

W-302

W-303

W-304

W-305

W-306

W-307

W-308

W-309

W-310

W-311

W-312

W-313

W-314

W-315

W-316

W-317

W-318

W-319

W-320

W-321

W-322

W-323

W-324

W-325

W-326

W-327

W-328

W-329

W-330

W-331

W-332

W-333

W-334

W-335

W-336

W-337

W-338

W-339

W-340

W-341

W-342

W-343

W-344

W-345

W-346

W-347

W-348

W-349

W-350

W-351

W-352

W-353

W-354

W-355

W-356

W-357

W-358

W-359

W-360

W-361

W-362

W-363

W-364

W-365

W-366

W-367

W-368

W-369

W-370

W-371

W-372

W-373

W-374

W-375

W-376

W-377

W-378

W-379

W-380

W-381

W-382

W-383

W-384

W-385

W-386

W-387

W-388

W-389

W-390

W-391

W-392

W-393

W-394

W-395

W-396

W-397

W-398

W-399

W-400

W-401

W-402

W-403

W-404

W-405

W-406

W-407

W-408

W-409

W-410

W-411

W-412

W-413

W-414

W-415

W-416

W-417

W-418

W-419

W-420

W-421

W-422

W-423

W-424

W-425

W-426

W-427

W-428

W-429

W-430

W-431

W-432

W-433

W-434

W-435

W-436

W-437

W-438

W-439

W-440

W-441

W-442

W-443

W-444

W-445

W-446

W-447

W-448

W-449

W-450

W-451

W-452

W-453

W-454

W-455

W-456

W-457

W-458

W-459

W-460

W-461

W-462

W-463

W-464

W-465

W-466

W-467

W-468

W-469

W-470

W-471

W-472

W-473

W-474

W-475

W-476

W-477

W-478

W-479

W-480

W-481

W-482

W-483

W-484

W-485

W-486

W-487

W-488

W-489

W-490

W-491

W-492

W-493

W-494

W-495

W-496

W-497

W-498

W-499

W-500

W-501

W-502

W-503

W-504

W-505

W-506

W-507

W-508

W-509

W-510

W-511

W-512

W-513

W-514

W-515

W-516

W-517

W-518

W-519

W-520

W-521

W-522

W-523

W-524

W-525

W-526

W-527

W-528

W-529

W-530

W-531

W-532

W-533

W-534

W-535

W-536

W-537

W-538

W-539

W-540

W-541

W-542

W-543

W-544

W-545

W-546

W-547

W-548

W-549

W-550

W-551

W-552

W-553

W-554

W-555

W-556

W-557

W-558

W-559

W-560

W-561

W-562

W-563

W-564

W-565

W-566

W-567

W-568

W-569

W-570

W-571

W-572

W-573

W-574

W-575

W-576

W-577

W-578

W-579

W-580

W-581

W-582

W-583

W-584

W-585

W-586

W-587

W-588

W-589

W-590

W-591

W-592

W-593

W-594

W-595

W-596

W-597

W-598

W-599

W-600

W-601

W-602

W-603

W-604

W-605

W-606

W-607

W-608

W-609

W-610

W-611

W-612

W-613

W-614

W-615

W-616

W-617

W-618

W-619

W-620

W-621

W-622

W-623

W-624

W-625

W-626

W-627

W-628

W-629

W-630

W-631

W-632

W-633

W-634

W-635

W-636

W-637

W-638

W-639

W-640

W-641

W-642

W-643

W-644

W-645

W-646

W-647

W-648

W-649

W-650

W-651

W-652

W-653

W-654

W-655

W-656

W-657

W-658

W-659

W-660

W-661

W-662

W-663

W-664

W-665

W-666

W-667

W-668

W-669

W-670

W-671

W-672

W-673

W-674

W-675

W-676

W-677

W-678

W-679

W-680

W-681

W-682

W-683

W-684

W-685

W-686

W-687

W-688

W-689

W-690

W-691

W-692

W-693

W-694

W-695

W-696

W-697

W-698

W-699

W-700

W-701

W-702

W-703

W-704

W-705

W-706

W-707

W-708

W-709

W-710

W-711

W-712

W-713

W-714

W-715

W-716

W-717

W-718

W-719

W-720

W-721

W-722

W-723

W-724

W-725

W-726

W-727

W-728

W-729

W-730

W-731

W-732

W-733

W-734

W-735

W-736

W-737

W-738

W-739

W-740

W-741

W-742

W-743

W-744

W-745

W-746

W-747

W-748

W-749

W-750

W-751

W-752

W-753

W-754

W-755

W-756

W-757

W-758

W-759

W-760

W-761

W-762

W-763

W-764

W-765

W-766

W-767

W-768

W-769

W-770

W-771

W-772

W-773

W-774

W-775

W-776

W-777

W-778

W-779

W-780

W-781

W-782

W-783

W-784

W-785

W-786

W-787

W-788

W-789

W-790

W-791

W-792

W-793

W-794

W-795

W-796

W-797

W-798

W-799

W-800

W-801

W-802

W-803

W-804

W-805

W-806

W-807

W-808

W-809

W-810

W-811

W-812

W-813

W-814

W-815

W-816

W-817

W-818

W-819

W-820

W-821

W-822

W-823

W-824

W-825

W-826

W-827

W-828

W-829

W-830

W-831

W-832

W-833

W-834

W-835

W-836

W-837

W-838

W-839

W-840

W-841

W-842

W-843

W-844

W-845

W-846

W-847

W-848

W-849

W-850

W-851

W-852

W-853

W-854

W-855

W-856

W-857

W-858

W-859

W-860

W-861

W-862

W-863

W-864

W-865

W-866

W-867

W-868

W-869

W-870

W-871

W-872

W-873

W-874

W-875

W-876

W-877

W-878

W-879

W-880

W-881

W-882

W-883

W-884

W-885

W-886

W-887

W-888

W-889

W-890

W-891

W-892

W-893

W-894

W-895

W-896

W-897

W-898

W-899

W-900

W-901

W-902

W-903

W-904

W-905

W-906

W-907

W-908

W-909

W-910

W-911

W-912

W-913

W-914

W-915

W-916

W-917

W-918

W-919

W-920

W-921

W-922

W-923

W-924

W-925

W-926

W-927

W-928

W-929

W-930

W-931

W-932

W-933

W-934

W-935

W-936

W-937

W-938

W-939

W-940

W-941

W-942

W-943

W-944

W-945

W-946

W-947

W-948

W-949

W-950

W-951

W-952

W-953

W-954

W-955

W-956

W-957

W-958

W-959

W-960

W-961

W-962

W-963

W-964

W-965

W-966

W-967

W-968

W-969

W-970

W-971

W-972

W-973

W-974

W-975

W-976

W-977

W-978

W-979

W-980

W-981

W-982

W-983

W-984

W-985

W-986

W-987

W-988

W-989

W-990

W-991

W-992

W-993

W-994

W-995

W-996

W-997

W-998

W-999

W-1000

W-1001

W-1002

W-1003

W-1004

W-1005

W-1006

W-1007

W-1008

W-1009

W-1010

W-1011

W-1012

W-1013

W-1014

W-1015

W-1016

W-1017

W-1018

W-1019

W-1020

W-1021

W-1022

W-1023

W-1024

W-1025

W-1026

W-1027

W-1028

W-1029

W-1030

W-1031

W-1032

W-1033

W-1034

W-1035

W-1036

W-1037

W-1038

W-1039

W-1040

W-1041

W-1042

W-1043

W-1044

W-1045

W-1046

W-1047

W-1048

W-1049

W-1050

W-1051

W-1052

W-1053

W-1054

W-1055

W-1056

W-1057

W-1058

W-1059

W-1060

W-1061

W-1062

W-1063

W-1064

W-1065

W-1066

W-1067

W-1068</

allowed for Location, RespPerson, and Comment. C is only used for TagNum, and if selected, the end-user must confirm the tag number in order to update the asset. This functionality is desired because very often, end-users do not check the barcodes on their items before replying. This can lead to incorrect data.

Email All: <input type="checkbox"/> P.I. <input type="checkbox"/> Resp Person <input type="checkbox"/> Other <input type="checkbox"/>				CC myself: <input checked="" type="checkbox"/>											
TagNum	Location	StandardName	Model	SerialNum	Manufacturer	PI	RespPerson	Other Contact	AcqDocNum	AcqDate	Comment	LastInvDate	AcqAmount		
0239485	1-190	SCANNER,OPTICAL	EYETRAC 210	9039-05-211	APPLIED SCIENTIFIC	HARRIS,WESLEY L	<input type="checkbox"/> YOUNG,LAURENCE RETMAN	<input checked="" type="checkbox"/> GIOVINE,GRAYSON	<input checked="" type="checkbox"/>	092232	03/05/1991	04/09/2004	7550.00		
0282648	1-110	SCANNER,OPTICAL	ODL150		CLARK INSTRUMENTATION	NELSON,KEITH A	<input type="checkbox"/> PAOLUCCI,DORA M	<input checked="" type="checkbox"/> GIOVINE,GRAYSON	<input checked="" type="checkbox"/>	487974	08/12/1994	10/17/2003	5930.00		
0297576	1-116	SCANNER,OPTICAL	M300	235148	HI TECH SCIENTIFIC	LIPPARD,STEPHEN	<input type="checkbox"/> VALENTINE,ANN M	<input checked="" type="checkbox"/> GIOVINE,GRAYSON	<input checked="" type="checkbox"/>	807758	10/13/1995	10/15/2003	25360.00		
0239485	37-146	SCANNER,OPTICAL	EYETRAC 210	9039-05-211	APPLIED SCIENTIFIC	HARRIS,WESLEY L	<input type="checkbox"/> YOUNG,LAURENCE RETMAN	<input type="checkbox"/>	<input checked="" type="checkbox"/>	092232	03/05/1991	04/09/2004	7550.00		
0239772	E15-044	SCANNER,OPTICAL	6350	911006	CAMBRIDGE TECHNOLOGY	NEGROPONTE,NICHO	<input type="checkbox"/> ST HILAIRE,PIERR	<input type="checkbox"/>	<input checked="" type="checkbox"/>	170000	03/19/1991	06/16/2004	1070.00		
0282648	2-083	SCANNER,OPTICAL	ODL150		CLARK INSTRUMENTATION	NELSON,KEITH A	<input type="checkbox"/> PAOLUCCI,DORA M	<input type="checkbox"/>	<input checked="" type="checkbox"/>	487974	08/12/1994	10/17/2003	5930.00		
0297576	18-444C	SCANNER,OPTICAL	M300	235148	HI TECH SCIENTIFIC	LIPPARD,STEPHEN	<input type="checkbox"/> VALENTINE,ANN M	<input type="checkbox"/>	<input checked="" type="checkbox"/>	807758	10/13/1995	10/15/2003	25360.00		

Email Selected

Figure 6-6: Emailing three items to Grayson Giovine.

Once the administrator decides how the fields will be displayed to the end-user(s), the emails can be sent. All of the items can be checked with the “Email All” box at the top, as shown in Figure 6-6. Alternately, individual items can be sent. In the figure, we check three items for “Other Contact”.

The “Other Contact” field exists because it is often discovered that the PI and responsible person in the asset database are not the best contacts for resolving the location of a particular item. Once the alternate contact is set for a particular item, it is stored in the “altcontacts” table of our MITDB database, and will appear the next time we seek to follow-up on this item.

To: ggiovine@mit.edu	Skip
Subject: MITDB Automated Report for Grayson Giovine	
Body:	
<p>You have assets that are currently unaccounted for. Click on the link to update their current location(s). Thank you.</p> <p>http://tagging.csail.mit.edu/propertyresp.php?confirmnum=2432b0685d7a6c18e4510d0bce</p>	
	Send Email

Figure 6-7: Confirmation dialog for sending an email generated for these items.

Once the “Email Selected” button is pressed, the items will be grouped and sent. A confirmation dialog will appear, as in Figure 6-7, and the emails will be sent once confirmed. The email has a nonce in it, which is tied to a set of assets.

6.3.4 Grouping Assets

Assets are grouped first by person, then by building and floor. The first grouping is fairly self-explanatory – in order to streamline the response process, we want to group as many items as possible to each person. If an end-user needed to click 20 different links to resolve his/her items, this would become tedious, and the user would likely give up. The second grouping is because we would like to display the floor contour for the user’s items, to facilitate quickly dragging the item icons to a new location. If there were multiple buildings and floor plans, this would become too complex and difficult to manage.

6.3.5 Confirming Updates

Once an end-user responds to the generated email, a pending update will be added to the database. When an administrator logs into the site, an “Updates Available” link will be available on the left sidebar, as shown in Figure 6-8. The administrator can then confirm or deny these updates.

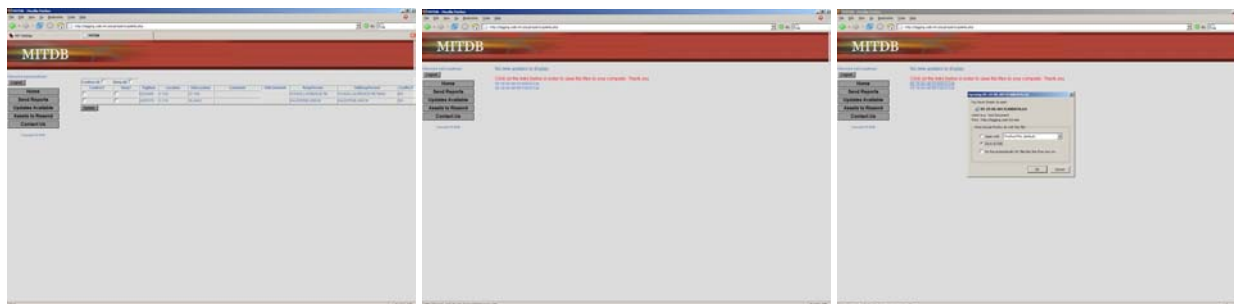


Figure 6-8: Displaying the updates from end-users (left); confirming these updates (middle), saving files (right).

If any updates are confirmed, two PRN files will be generated. The file names conform to the SumProp convention: <date><AM/PM> <SCANDATA/RETGDATA>.txt. The RETGDATA file is always blank, but we provide it here because of the fact that SumProp requires both files be named correctly. SCANDATA contains information for the items we have updated, and conforms to the format SumProp expects.

6.3.6 Following-up With Users

If the items that were sent to a particular person are unresolved after 3 weeks (or any amount of time the administrator specifies), an “Assets to Resend” link will appear on the left sidebar when

an administrator logs into the tool. He may then choose to allow or disallow this follow-up email to be sent. When the follow-up is sent, the notice number will be included (e.g. (2nd Notice), (3rd Notice), etc.).

6.4 End-User Interface Design

This section will detail the functionality of the end-user side of the Property Tool. In designing this, we wanted it be user-friendly. If this were not achieved, users would either entirely ignore updating their items, or just use email to do so. A second goal is that we wanted to assure that end-users would never be modifying real data. Therefore, each time an asset is followed-up on, it is copied to the “assets_temp” table of our database. Any changes an end-user makes only affects this table, and the “assets” table will not be updated until an administrator confirms updates.

6.4.1 Visualizations

Figure 6-8 shows the display that appears when a user clicks the email shown in Figure 6-7.

MITDB - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://tagging.csail.mit.edu/propertyresp.php?confirmnum=2432b0685d7a6c18e4510d0bcd222904

MITDB

Username:
 Password:

Copyright © 2006

Please update the locations of the items listed. You may either move the pushpins to their proper locations, or type into the text boxes.

If a particular item has a text box for its tag number, you *must* enter the correct tag number for that item for it to validate correctly.
 Note: If the item is not easily visible (e.g. in the left hand cabinet), please note that in the 'Comment' field.

Press update to propagate changes to the Property Office.

Search for Location

 e.g. "10-250" or "Building 1" or "Stata Center"

MIT Map MIT Aerial Hybrid

Old Locations: 1-110, 1-190, 1-116

TagNum	Location	RespPerson	StandardName	Model	SerialNum	Manufacturer	AcqDocNum	AcqDate	AcqAmount	Comment
1-110		PAOLUCCI,DORA M	SCANNER,OPTICAL	DDL150		CLARK INSTRUMENTATION	487974	08/12/1994	\$5930.00	
1-190		YOUNG,LAURENCE RETMAN	SCANNER,OPTICAL	EYETRAC 210	9039-05-211	APPLIED SCIENTIFIC	092232	03/05/1991	\$7550.00	
1-116		VALENTINE,ANN M	SCANNER,OPTICAL	M300	235148	HI TECH SCIENTIFIC	607758	10/13/1995	\$25360.00	

If you do not know where the items are and would like to pass them on to someone else, please do so below.

Email Address:

Have comments about this tool? Send us feedback!

If you would like to correspond about these items with the Property office, please do so below:

Figure 6-8: Property Tool from the end-user side - visualizing 3 items in building 1, floor 1.

Since all of the items displayed are on the first floor of building 1, the contour for this building displays. The user can interact with this in three ways – editing the table below the map, dragging the item icons to their correct locations, or clicking on an icon and changing the location in the marker bubble. In this example, the user is required to confirm the tag numbers of the items before sending – this can be done in the table or the marker bubble.

6.4.2 Responding To Inquiries

When presented with this interface, our user has three choices for how to respond:

- 1) Update the locations of the items (if necessary), confirm the tag numbers, and press the “Update” button.
- 2) Pass items along to someone else using the “Pass Along” button.
- 3) Send an email directly to the Property Office with a form.

Option #1 is the most desired option, because it directly addresses the problem in an automated fashion. Option #2 is used for situations where a particular user does not know the whereabouts of the items, and so passes the task onto someone else. This may happen in many situations – to give an example, a professor may purchase laptops for students in a research group, and may not know where the laptops currently reside. The professor would use this option so that the students could respond to the Property Office. Option #3 is the least desired option, as it is the equivalent of an email (which is the old Property Office medium for follow-ups). However, this option is desired for cases that are more complex or where more information is desired.

We also provide a feedback form in order to acquire useful comments about the tool.

6.5 User Testing

User testing played a role in the Property Tool’s evolution, because in the end, it was meant to save time for its user base. This section outlines our user testing procedure and results.

6.5.1 Procedure

The user testing of this system was done on an informal basis. After completing each version of the program, it was sent to a member of the Property Office for evaluation. Through a continuous feedback loop, the program was improved and features were added to help meet the needs of the Property office.

6.5.2 Results

Paul MacDonald made the following comments about the Property Tool:

- 1) Eliminates the step of manually typing pertinent information for each item by supplying all necessary fields from the captured PRN file.
- 2) Helps to eliminate the added step of searching the online directory for the email addresses of recipients.
- 3) Very user friendly, with a screen that is clear, concise and easy to follow (although a tutorial for the uninitiated would be needed).
- 4) Very flexible – all field options as well as all email options are easily added or subtracted to suit specific needs.
- 5) Presents the information within the email in an organized and professional manner.

Based on these comments, the tool certainly seems to be a desirable one, and something that will save time for those in the Property Office.

Chapter 7

Other Applications / Visualizations

This chapter presents other applications and visualizations that have been created using the framework proposed in Chapter 4. We will discuss a visualization builder for easily visualizing different data sets, and a tool for binding icons to category names.

7.1 Visualization Builder

In the process of developing the WikiMap, it became apparent that the marker bubble (which pops up upon clicking an object) could be streamlined. Whenever a minor fix needed to be made to the text or behavior of the bubble, it was a manual change that sometimes took a bit of time to get correct. The solution to this is a simple visualization builder.

Using this method, we can easily visualize objects from any data set that we have federated. Figures 7-1 and 7-2 demonstrate how we can create a simple display for Equipment objects. Figure 7-2 shows the set of commands that leads to the bubble displayed in Figure 7-1.



Figure 7-1: Equipment object marker bubble.

EquipObj									
Update									
ID	test	variable	type	colspan	align	height	row	col	grouping
ItemLocation		location	div	1	center	0	1	1	d_Array_Row
ItemDescription		description	div	1	center	0	2	1	d_Array_Row
ItemManufacturer		manufacturer	div	1	center	0	3	1	d_Array_Row
ItemModel		modelnumber	div	1	center	0	4	1	d_Array_Row

Figure 7-2: Visualization information for Equipment data.

Figures 7-3 and 7-4 demonstrate the same visualization for Schedules data and People data, respectively.



Figure 7-3: Schedules object marker bubble (left); visualization info for Schedules data (right).

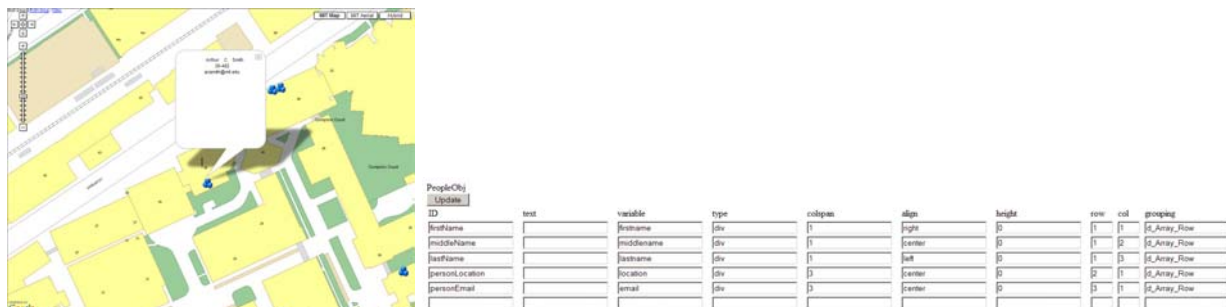


Figure 7-4: People object marker bubble (left); visualization info for People data (right).

These three visualizations are fairly simple. For the most part it just involves the display of a few variables and text. What if we wanted to do something more complicated? For example, the MIT WikiMap bubble is highly interactive and complex. Figure 7-5 demonstrates the two main states for the bubble – display mode and edit mode.



Figure 7-5: WikiMap object marker bubble – in display mode (left) and edit mode (right).

We can see how this marker bubble is produced in Figure 7-6. This figure includes both simple fields as well as complex, user-defined fields. This will be explained further in section 7.1.2.

WikiObj									
Update									
ID	text	variable	type	colspan	align	height	row	col	grouping
ItemName		name	div	2	center	0	1	1	d_Array_Row
ItemLocation		location	div	2	center	0	2	1	d_Array_Row
ItemCategory		category	div	2	center	20	3	1	d_Array_Row
ItemDescription		description	div	2	left	110	4	1	d_Array_Row
		requestDeletion	1			0	5	1	d_Array_Row
		editData	1			0	5	2	d_Array_Row
	Item Name		1			0	1	1	e_Array_Row
ItemNameE		nameE	inputtext	1		0	1	2	e_Array_Row
	Location		1			0	2	1	e_Array_Row
ItemLocationE		locationE	div	1		0	2	2	e_Array_Row
			1			0	3	1	e_Array_Row
	Drag icon to change		1			0	3	2	e_Array_Row
	Category		1			0	4	1	e_Array_Row
ItemCategoryE		categoryE	selectCategory	1		0	4	2	e_Array_Row
	Description		2			0	5	1	e_Array_Row
ItemDescriptionE		descriptionE	textarea	2		0	6	1	e_Array_Row
		backLocation	1			0	7	1	e_Array_Row
		saveCancelData	1		right	0	7	2	e_Array_Row

Figure 7-6: Visualization information for WikiMap data.

As part of the interface for changing an object type’s marker bubble, the last row is blank – to facilitate easily adding new information. When “Update” is pressed, these settings are stored in the “marker_display” table of our database. These settings are loaded when an object of a particular type is displayed on the WikiMap.

7.1.1 XML Representation

When the settings are being loading by the WikiMap, some of the fields may contain variables. These variables must be contained in the XML output for that object, or else “undefined” will be returned.

```

- <xml>
  <type name="equipment">
    <secondtype name="description">
      + <record id="EquipObj000000032120"></record>
      + <record id="EquipObj000000032156"></record>
      + <record id="EquipObj000000027986"></record>
      + <record id="EquipObj000000032143"></record>
      + <record id="EquipObj000000017355"></record>
      + <record id="EquipObj000000026094"></record>
      + <record id="EquipObj000000030366"></record>
      + <record id="EquipObj000000026188"></record>
      + <record id="EquipObj000000020810"></record>
      + <record id="EquipObj000000030814"></record>
      + <record id="EquipObj000000030872">
        <location>35.122</location>
        <gpoint>709582.1875000</gpoint>
        <gpoint>495924.9375000</gpoint>
        <manufacturer>NA</manufacturer>
        <modelnumber>INSP DATE 6-84</modelnumber>
        <description>FIRE EXTINGUISHER</description>
      </record>
      ...
    </xml>
  </xml>

- <xml>
  <type name="people">
    <secondtype name="name">
      + <record id="People19755"></record>
      + <record id="People19740"></record>
      + <record id="People19767"></record>
      + <record id="People19753"></record>
      + <record id="People19766"></record>
      + <record id="People19737"></record>
      + <record id="People19782"></record>
      - <record id="People19704">
        <location>38.482</location>
        <gpoint>710040.7500000</gpoint>
        <gpoint>496156.3125000</gpoint>
        <firstname>Arthur</firstname>
        <middlename>C</middlename>
        <lastname>Smith</lastname>
        <email>acsmith@mit.edu</email>
      </record>
      ...
    </xml>
  </xml>

- <xml>
  <type name="wiki">
    + <record id="00fa015816c83ae51691bfa9dd5f45">
      <name>Bosworth's Cafe</name>
      <nameE>Bosworth's Cafe</nameE>
      <location>7-100 lobby</location>
      <locationE>7-100 lobby</locationE>
      <gpoint>709864</gpoint>
      <gpointE>709864</gpointE>
      <gpoint>495522</gpoint>
      <gpointE>495522</gpointE>
      <category>Quick Food</category>
      <categoryE>Quick Food</categoryE>
      <imageLoc>
      <imageLocE>
      <description>Place to get a coffee before classes!</description>
      <descriptionE>Place to get a coffee before classes!</descriptionE>
      <displayed>true</displayed>
      <editable>true</editable>
      <isTack>false</isTack>
      <recent>false</recent>
    </record>
    ...
  </xml>

```

Figure 7-7: XML output for Equipment, Schedules, People, and Wiki objects, respectively.

The figure above shows examples of XML output for our objects. The first three are self-explanatory. In the Wiki object XML, we can see that some items are repeated. “name” and “nameE” have the same information, and similarly many other tags are duplicated (with just an “E” at the end). This is because the marker bubble for Wiki objects refers to variables in both display mode and edit mode. Therefore, though we could save bandwidth by having some other convention, we do this for simplicity reasons. Any time a variable is referenced in the marker bubble construction, it must be contained in the XML output.

7.1.2 Visualization Types

This will be described in more detail in Appendix B, API Documentation. However, we will briefly discuss it here.

This tool for generating marker bubbles is a table constructor. It uses rows and columns to determine where to place information. A particular entry can span multiple columns, have its height defined in pixels, and have its alignment set. Entries are either a variable, text, or user-defined.

As indicated previously, if a variable is specified it must exist in the XML output for the object. The main types for displaying variables are “div”, “inputtext” (input text box), and “textarea”. This simply determines how the variable will be displayed. For Wiki objects, the display mode uses “div” for all of its variables, while edit mode will sometimes use “inputtext” or “textarea” in addition to “div”.

For text, the type is blank. The desired text can then simply be inserted in the “text” column (see Figure 7-6).

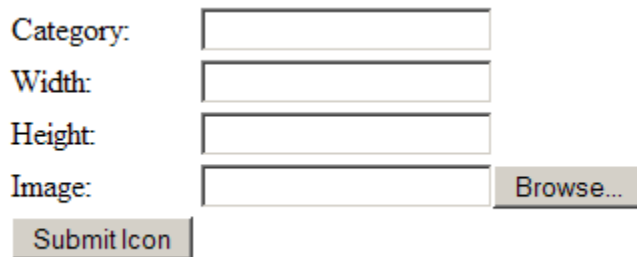
User-defined types were needed to add the complexity necessary for Wiki objects. If the “Edit” button is clicked while in display mode, edit mode will appear. In order to do this, there must be custom JavaScript code that runs when the button is clicked. Therefore, user-defined types allow administrators to tie blocks of HTML and JavaScript to a type, and whenever that type is placed in the marker bubble settings, this will be used.

Finally, we will explain the “grouping” column. For Wiki objects, we see there are two groupings – “d_Array_Row” and “e_Array_Row”. For Equipment, Schedules, and People, there is only one grouping. This simply corresponds to how many distinct tables should be created. With Wiki objects, the marker bubble requires two tables for the two modes.

7.2 Category Icon Tool

The Category Icon Tool binds images to category names. Once a particular binding is contained in the database, it will be loaded when the MIT WikiMap is accessed. All of the category bindings are output as GIcons with the necessary properties. That category may then be added to the map using the following call:

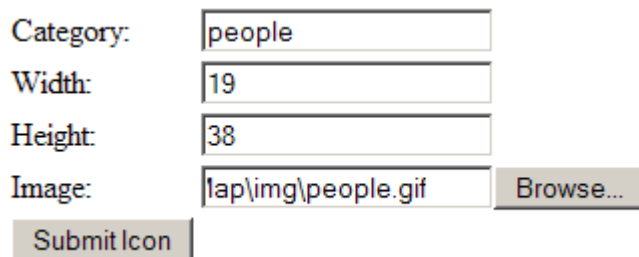
```
createGMarker(gpoint,category)
```



A screenshot of the Category Icon Tool web form. It contains four input fields: 'Category:', 'Width:', 'Height:', and 'Image:'. To the right of the 'Image:' field is a 'Browse...' button. Below the 'Image:' field is a 'Submit Icon' button.

Figure 7-8: Category Icon Tool – simple tool that allows users to bind an image to a category; this can later be used in the WikiMap application (or others).

This is a clear example where putting in an additional level of abstraction helped simplify the system. To give an example, we can add a new icon for the “People” category of objects. We simply go to <http://wikimap.csail.mit.edu/createicon.php>, then enter the appropriate fields.



A screenshot of the Category Icon Tool web form with example data entered. The 'Category:' field contains 'people', 'Width:' contains '19', 'Height:' contains '38', and 'Image:' contains 'lap/img/people.gif'. The 'Browse...' button is to the right of the 'Image:' field. The 'Submit Icon' button is at the bottom.

Figure 7-9: Adding an image for the “people” category.

Once this is added, so long as the returned XML for a people object has “category” set, we are able to display the image that is now bound to this category.

```

- <record id="People19704">
  <category>people</category>
  <location>38-482</location>
  <gpointx>710040.7500000</gpointx>
  <gpointy>496156.3125000</gpointy>
  <firstname>Arthur</firstname>
  <middlename>C</middlename>
  <lastname>Smith</lastname>
  <email>acsmith@mit.edu</email>
</record>

```

Figure 7-10: XML representation for people object.

This provides a nice way for users to add interesting icons to the infrastructure. The more icons we have, the better the visualizations we can create. A visualization of multiple object types provides less information if the multiple types use the same default icon.



Figure 7-11: New icon being displayed in MIT WikiMap.

7.3 Evacuation Routes

This section will describe how evacuation routes are generated using schedules and hazardous materials data.

<More on this to be inserted later>

Chapter 8

Conclusions & Future Work

This chapter discusses conclusions we can draw and future work that can be done in this realm.

8.1 Shortcomings

There are shortcomings with this project that can be improved in the future. The first that becomes apparent is the difficulty making these applications cross-browser compatible. Whereas Google has an entire team of programmers to work on their map, it is much more difficult to sufficiently test and debug these issues with a small team. Therefore, while our suite of applications run nicely using the Mozilla Firefox browser, there is less than full functionality with other browsers. We have done testing in Internet Explorer and have achieved close to full functionality, though there are a few small issues to resolve.

A second shortcoming is simply that we have not had enough time to deploy and test this system with a full user base. Doing so will bring to the fore new issues that perhaps have not been anticipated. This deployment will occur shortly, however, so a future project may be able to address some of the issues that arise.

8.2 Future Work

This section will discuss various improvements that can be made to the applications already built, as well as some future work that would prove useful.

8.2.1 MIT WikiMap Improvements

As mentioned in the shortcomings section, cross-browser compatibility is something we would like to achieve. At the very least, we would like to be nearly bug-free in Mozilla and Internet Explorer. If possible, we would also like to cover other browsers. Though it is difficult to find reliable statistics on Internet browser use [14], estimates indicate that approximately 11% of people use Mozilla, Firefox, or a Gecko-based browser. Approximately 83% use Internet Explorer [31]. These figures vary widely from site to site – for example, Figure 8-1 shows

browser statistics for web.mit.edu. Approximately 70% use IE, 20% Mozilla/Firefox, and 10% use another browser.

Therefore, we have achieved 90+% coverage of our user base if we support these two browsers. While it is important to get near 100% coverage, this is the subject of future work, rather than something that needs to be achieved immediately.

		200503		Total
		March 2005		
Browser	Browser Version	MITNet	Non MITNet	
Firefox		94,845	194,861	279,706
Lynx			2	2
Microsoft Internet Explorer	2		27	27
	3	1	21	22
	4	108	1,022	1,130
	5	12,259	81,193	93,452
	6	315,582	1,177,996	1,493,578
	7	6	7	13
Mozilla		190	216	406
	0	48	816	864
1	78,180	83,568	159,748	
Netscape		1		1
	(4	4
	3	5	43	48
	4	1,280	4,230	5,510
	5	2		2
	6	2,762	1,619	4,381
7	104,743	51,756	156,499	
Opera		1,897	11,303	13,000
Other		752	3,482	4,234
Safari		14,688	35,207	49,895
Total		825,149	1,837,381	2,262,530

Figure 8-1: Browser statistics for <http://web.mit.edu>.

There are a few features we would like to add to the MIT WikiMap as well. First, we would like to add the ability to upload and share images. This is useful both because it would make the WikiMap more enjoyable from a user standpoint, but also because we would then be able to use these images to create landmarks in route navigation. Therefore, if someone takes a picture of the entrance to 77 Massachusetts Avenue from the steps of the Student Center, we could ask the user to provide the camera location and viewing angle. A route going through this location would then display this image at the place where it was observed, thus creating a landmark along the route.

We would also like to add the ability to personalize the WikiMap. If a student wanted to map his/her classes, but did not want this viewable to the entire MIT community, we could allow for restricted access to Wiki objects.

Finally, we would like to expand the wiki portion of the WikiMap. We would like to allow users to view how entries have changed over time. We also might like to allow for extended entries – currently the bubble that pops up for each object is too small to display more than 50 words in the description. We can experiment with using the sidebar to give more detailed descriptions of objects.

8.2.2 Application Builders

It may be useful to harness all the components we have created in some type of application builder. This builder would make it easy to build custom applications of this flavor, and would dramatically reduce the workload of data set administrators for creating visualizations of their data. It has yet to be determined how feasible doing this is, though it could be the subject of a future project.

8.2.3 Advanced Visualizations

We currently have the capability to visualize discrete objects on a map, and can produce relatively simply colorings of the map. These colorings are implemented using the GPolyline constructor. Though this is nice in that it fits into our current framework, it is not easy to create more advanced visualizations using this method.

In order to achieve advanced visualizations, such as a color gradient to show the price per unit area of the assets on campus, we would need something more sophisticated. We could implement this by generating images with a color gradient and layering them on top of the map.

Furthermore, 3-D visualizations are something we would like to achieve. This may not be feasible using a JavaScript-based implementation, but the data model can be used to code other implementations that are more conducive to 3-D modeling.

8.3 Conclusions

The implications of building a system for federating data and automating incorporation into a map context are far-reaching. This system could not only be used to integrate distinct data sources at MIT, but it could also be used on other campuses or locations generally. We now have a system and API in place that can be rolled out and used in a variety of contexts.

Bibliography

- [1] Agrawala, M. and Stolte, C. *Rendering Effective Route Maps: Improving Usability Through Generalization*. In Proc. of SIGGRAPH 2001, pp. 241-249, 2001.
- [2] Bedrosian, Peter, et al. *Classroom Scheduling and Management*. Massachusetts Institute of Technology Office of the Registrar. Spring 2006. <<http://web.mit.edu/registrar/www/schedules/>>
- [3] Bedrosian, Peter, et al. *Subject Schedules by Section*. Massachusetts Institute of Technology Office of the Registrar. Spring 2006. <<https://student.mit.edu/cgi-bin/sfrresch.sh>>
- [4] Battat, Jonathan. *A Robust Geospatial Data-Modeling Framework and Interface for Navigation Systems*. Massachusetts Institute of Technology, 2005.
- [5] Bourke, Paul. "Determining if a Point Lies on the Interior of a Polygon." November 1987. Centre for Astrophysics and Supercomputing, Swinburne University of Technology. Spring 2006. <<http://astronomy.swin.edu.au/~pbourke/geometry/insidepoly/>>
- [6] *CSAIL Event Calendar*. Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory. Spring 2006. <<http://www.csail.mit.edu/events/eventcalendar/calendar.php>>
- [7] "Database speed." The Particle Revelation. Brooklyn College Database Systems class. Fall 2005. <<http://www.theparticle.com/cs/bc/dbsys/speed.pdf>>
- [8] Dean, Denis J. "The State Plane Coordinate System." Geodesy, Cartography and Map Reading, College of Natural Resources at Colorado State University. Spring 2006. <http://www.warnercnr.colostate.edu/class_info/nr502/lg3/datums_coordinates/spcs.html>
- [9] "Finding Data for Maps." *Geographic Information Systems*. Amherst College Information Technology. Spring 2006. <<http://www.amherst.edu/it/software/gis/gis-finding-data/>>
- [10] "Geographic Coordinate System." *Wikipedia: The Free Encyclopedia*. May 2006. Spring 2006. <http://en.wikipedia.org/wiki/Geographic_coordinates>
- [11] Giovine, Grayson. *Location-Aware Asset Tagging Using Crickets*. Massachusetts Institute of Technology, 2005.
- [12] Giovine, G., Stoltzman, W. and Whiting, E. *MIT WikiMap*. 6.831: User Interface Design & Implementation, Final Report, Massachusetts Institute of Technology, December 2005.

- [13] Goodchild, Michael F. "Spatial Analysis and Geographic Information Systems." 2001 ESRI User Conference. National Center for Geographic Information and Analysis. University of California, Santa Barbara. Spring 2006. <http://www.csiss.org/learning_resources/content/good_sa/>
- [14] Hayden, Lance. "Debunking Browser Stats." All Things Web. Spring 2006. <<http://www.pantos.org/atw/f-35448.html>>
- [15] Hoffman, Gernot. "Distance Between Line Segments." University of Applied Sciences Oldenburg/Ostfriesland/Wilhelmshaven. Spring 2006. <<http://www.fho-emden.de/~hoffmann/xsegdist03072004.pdf>>
- [16] Holovaty, Adrian. *Chicagocrime.org*. Wilson Miner. Chicago Police Department Citizen ICAM. Fall 2005. < <http://www.chicagocrime.org/>>
- [17] Katz, Boris, et al. "Omnibase: Uniform Access to Heterogeneous Data for Question Answering." June 2002. Massachusetts Institute of Technology Artificial Intelligence Laboratory. Spring 2006. <<http://groups.csail.mit.edu/infolab/publications/Katz-et-al-NLDB02.pdf>>
- [18] Kulikov, V. *Generating a Model of the MIT Campus Terrain*. M.Eng. Thesis, Massachusetts Institute of Technology, May 2004.
- [19] Lischinski, D. *Incremental Delaunay Triangulation*. In Heckbert, S. (ed.), *Graphics Gems IV*, Academic Press, 1994.
- [20] Matsakis, Nick. *CSAIL Event Calendar, RSS Information*. Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory, Learning and Intelligent Systems. PHP iCalendar. Spring 2006. <<http://lis.csail.mit.edu/csailvts/rss/index.php>>
- [21] McGehee, Brad M. et al. "Tips on Performance Tuning SQL Server Indexes." Spring 2006. <http://www.sql-server-performance.com/optimizing_indexes.asp>
- [22] *MIT campus map home*. Massachusetts Institute of Technology Department of Facilities and Information Services & Technology. Fall 2005. <<http://whereis.mit.edu/map-jpg>>
- [23] *MIT Data Warehouse*. Massachusetts Institute of Technology Information Services and Technology. Fall 2005. <<http://web.mit.edu/warehouse/>>
- [24] *MIT Facilities—Maps and Floor Plans*. Massachusetts Institute of Technology Department of Facilities. Spring 2006. <<https://floorplans.mit.edu/>>
- [25] *MIT Property*. Massachusetts Institute of Technology Property Office. Fall 2005. <<http://controllers.mit.edu/property>>

- [26] Nichols, Patrick James II. *Location-Aware Active Signage*. January 2004. Massachusetts Institute of Technology. Spring 2006. <http://groups.csail.mit.edu/graphics/pubs/thesis_nichols.pdf>
- [27] Pfost, Donald. "Precision Agriculture: Global Positioning System (GPS)." January 2006. University of Missouri Extension. Spring 2006. <<http://extension.missouri.edu/explore/envqual/wq0452.htm>>
- [28] Rademacher, Paul. *HousingMaps*. Fall 2005. <<http://www.housingmaps.com/>>
- [29] *ShuttleTrack*. Massachusetts Institute of Technology Parking Office. Fall 2005. <<http://shuttletrack.mit.edu>>
- [30] Sunday, Dan. "The Crossing Number". *Fast Winding Number Inclusion of a Point in a Polygon*. Spring 2006. <http://softsurfer.com/Archive/algorithm_0103/algorithm_0103.htm#Crossing%20Number>
- [31] Upsdell, Chuck. "Trends in Browsers, Colour-Depths, and Resolutions." May 2006. Browser News. Spring 2006. <http://www.upsdell.com/BrowserNews/stat_trends.htm>

Appendix A

Project Build Instructions

This appendix describes how to build and run the scripts and applications that are part of this thesis. Some of these can be run anonymously; others require membership in the BMG group. These instructions assume a LINUX command line environment.

A.1 Checkout Instructions

The applications and scripts described in this thesis are contained in the CSAIL SVN repository as part of the walkthrough/mit source tree. There are two paths to take:

- 1) Checking out the entire directory, necessary to running the pipeline and all the applications/scripts in this thesis
- 2) Checking out just the federation folder – to run everything in this thesis with the exception of the pipeline/dangling portal connector.

In order to check out the entire directory, type the following command for a CSAIL machine with local access:

```
$ svn checkout file:///afs/csail/group/rvsn/Repository/walkthru/mit/src
```

For remote access via ssh, type:

```
$ svn checkout svn+ssh://login.csail.mit.edu/afs/csail/group/rvsn/Repository/walkthru/mit/src
```

In order to check out just the federation folder, add “/federation” to the end of each of the commands above.

In addition to checking out all the code for this project, it is also necessary to have Apache, PHP, and MySQL running on the server to which we check out the files. Due to the differences in setting up these three on different platforms, it would not be useful to create a script that attempts to do all this automatically. To download a tar file for each of these, go to the links below:

Apache: <http://httpd.apache.org/>

PHP: <http://www.php.net/downloads.php>

MySQL: <http://dev.mysql.com/downloads/>

After downloading each of these, we compile and install them. When compiling PHP, make sure to enable PEAR. This is a library that enables our PHP scripts to connect to any type of database in a generalized manner.

All of this has already been done on agenda.csail.mit.edu. Therefore, if we are checking out to a folder here, installation of Apache/PHP/MySQL is unnecessary.

A.2 Build & Run Instructions

This section describes how to build and run the code for this project. It is broken into three parts: pipeline, database, and applications.

A.2.1 Pipeline

After checking out the entire directory, move to where the local checkout resides, and type:

```
$ ./pipeline.py
```

This will run the pipeline described in Chapter 3 in its entirety.

A.2.2 Database

In order to create the MITDB database and populate the tables within it, we first move to the federation folder, then run a single script:

```
$ php -f createAndPopulateMITDB.php
```

This script is simply the combination of four scripts run in order. To run these individually, we run:

```
$ php -f scripts/createTables.php  
$ php -f ParseFunctions/populateSpatialData.php  
$ php -f ParseFunctions/parseSpaceExtents.php  
$ php -f ParseFunctions/federateAllDataSets.php
```

The “createTables.php” script will create the MITDB database and the necessary tables. The “populateSpatialData.php” script will populate the spatial tables with the data from the data

model. “parseSpaceExtents.php” will take all the space extents from the data model and place them in the “rooms” table of the database. “federateAllDataSets.php” will federate all the data sets we have discussed in this thesis. For data sets that are being federated via a file, all the most recent files are in the directory with this script. In order to use a more recent copy, we can simply replace the file.

A.2.1 Applications

The applications described in this thesis can be run using a vanilla web browser (with JavaScript enabled). In order to access them, place the following directories in a location where they are web accessible:

/federation/WikiMap/
/federation/PropertyTool/

After doing this, we can run the MIT WikiMap at “/federation/WikiMap/mitwikimap.php”, and the Property Tool at “/federation/PropertyTool/index.php”. To access the administrative side of the Property Tool, go to “/federation/PropertyTool/property.php”. To access the end-user side, go to “/federation/PropertyTool/property.php”. To access the category icon tool, go to “/federation/WikiMap/createicon.php”.

Appendix B

API Documentation

This appendix discusses the API we have specified for federating data and bringing it into a map context.

<More on this to be inserted later>

Appendix C

Permalinks

This appendix provides permalinks that can be followed to demonstrate visualizations produced using this infrastructure.

Display of 10-250 room contour:

<http://wikimap.csail.mit.edu/?query=10-250>

Display of 10-2 floorplan:

<http://wikimap.csail.mit.edu/?x=710127.522345&y=495696.634022&zoom=9&query=10-2>

Display of Building 1, providing center and zoom:

<http://wikimap.csail.mit.edu/?x=710118.408478&y=495713.563832&zoom=9>

Display of Building 1:

<http://wikimap.csail.mit.edu/?x=710077.0705754575&y=495120.54374368454&zoom=9&query=Building%201>

Display of Wiki object – W20 cluster:

<http://wikimap.csail.mit.edu/?x=709453&y=495543&zoom=9&query=W20%20cluster>

Display of 6.170 classrooms:

<http://wikimap.csail.mit.edu/?x=709954.1875&y=495240.9375&zoom=10&query=6.170>

Display of open classes on Tuesday in building 1 between 11 AM and 12 PM:

<http://wikimap.csail.mit.edu/?x=710139.875&y=495036.28125&zoom=9&query=open%20between%2011%20and%2012%20on%20tuesday%20in%201>

Display of people with last name “Hunter”:

<http://wikimap.csail.mit.edu/?x=712328.25&y=496026.3125&zoom=11&query=hunter>

Appendix D

Contact Information for Data Sets

This appendix provides contact information for the data sets that either have been federated or that we are aiming to federate in the near future.

Assets: Paul MacDonald, paulmac@mit.edu
Michael F. McCarthy, mmccarth@MIT.EDU

EHS: Bill Van Schalkwyk, billv@MIT.EDU
Laurie Veal, veal@MIT.EDU

Equipment: Michael Sherman, MSherman@PLANT.MIT.EDU
Greg Knight, GKnight@PLANT.MIT.EDU

People: Maija Ahlquist (Data Warehouse), maija@MIT.EDU

Police: John E. Driscoll, driscol@MIT.EDU

Schedules: Wayne B. Johnson, wayneb@MIT.EDU

ShuttleTrack: Tom Coveney, lensman@MIT.EDU

Appendix E

Algorithms

This appendix details algorithms that were used in this thesis. These algorithms are not new material, and so did not require explanation in the body of the thesis.

E.1 Point-in-Polygon Test

We use the crossing number test to determine whether a point is in a particular polygon. This method counts the number of times a ray extending from the point crosses the edges of the polygon. If this number is even, we're outside the polygon; if it's odd, we're inside. This can be explained intuitively because every time we cross an edge of the polygon, we've either gone from inside to outside, or outside to inside. Since the ray ends up outside the polygon eventually, we can see how this rule holds.

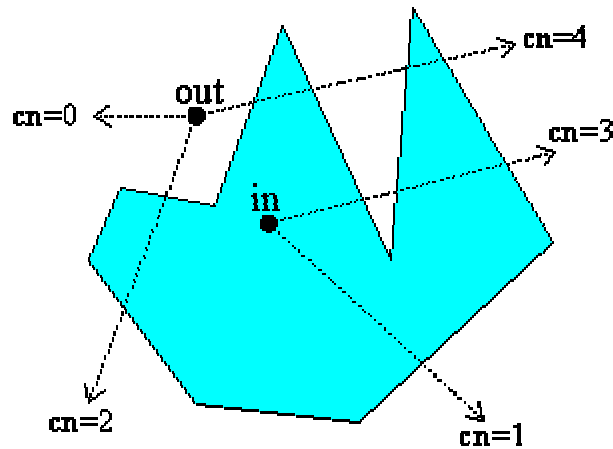


Figure C-1: Rays for sample points that demonstrate the technique [30].

We must also decide whether we consider a point on the polygon boundary to be inside or outside. Typically, the convention is that a point on a left or bottom edge is inside, and a point on a right or top edge is outside. For our purposes, however, we want to provide the user with the freedom to choose either space if a tack is placed on a boundary. Therefore, we always consider a point on the boundary to be inside.

In our test, we draw a horizontal ray to the right of our point, and count the crossings. In doing this count, we obey two rules:

- 1) Horizontal edges are excluded
- 2) The intersection of the edge and ray must be to the right of our point $P(x,y)$

Figure C-2 shows the actual code that does the point-in-polygon test. This code was adapted from C code that was found on a website [5]. It is commented enough that it should be understandable. The only point to note is that \$arrayPoints is an array of X, Y coordinates, each represented by an array of length 2. X corresponds to 0 in this array, and Y corresponds to 1.

```
function pointInPolygonTest($x,$y,$arrayPoints) {
    // the crossing number counter
    $counter = 0;

    // loop through all edges of the polygon
    for ($i=1;$i<count($arrayPoints);$i++) {
        //check if y is between the min and max of the edge, and x <= than the max
        if ($y >= min($arrayPoints[$i-1][1],$arrayPoints[$i][1])) {
            if ($y <= max($arrayPoints[$i-1][1],$arrayPoints[$i][1])) {
                if ($x <= max($arrayPoints[$i-1][0],$arrayPoints[$i][0])) {
                    //check that the edge is not horizontal
                    if ($arrayPoints[$i-1][1] != $arrayPoints[$i][1]) {
                        //calculate the slope
                        $slope = ($arrayPoints[$i][1] - $arrayPoints[$i-1][1]) / ($arrayPoints[$i][0] - $arrayPoints[$i-1][0]);

                        //calculate the x intersection point of the horizontal ray with the edge
                        $xInter = ($y - $arrayPoints[$i-1][1]) / $slope + $arrayPoints[$i-1][0];
                        //if the edge is vertical, or the intersection is to the right of the point,
                        //we increment the counter
                        if (($arrayPoints[$i-1][0] == $arrayPoints[$i][0]) || ($x <= $xInter)) {
                            $counter++;
                        }
                    }
                }
            }
        }
    }

    if (($counter % 2) == 1)
        return true;
    else
        return false;
}
```

Figure C-2: Code for point-in-polygon test.