IPRO 305
The Applications of Pervasive Computing

Fall 2004 Final Report

1. BACKGROUND

Pervasive computing is a broad new research field in computer science. Whereas traditional interfaces with computers have been through a keyboard and monitor, pervasive computing seeks to weave computers seamlessly into our daily lives. In a world with pervasive computing you'll be able to go shopping and automatically get more information about a product that is tailored to your preferences. You'll be able to open your notebook computer up and automatically discover network services that are present in your area. You'll be able to start complex computing jobs from your handheld device and not have to worry about where the results are going to be stored. These are just a few of the things that will come as a result of living in a truly pervasive world.

2. OBJECTIVES

The fall 2004 IPRO 305 team will be continuing the efforts of ongoing research at IIT in the realm of Pervasive Computing. The team will be focusing on the challenge of integrating Pervasive Computing into a specific aspect of the IIT campus. The overall vision of the IPRO 305 team is to continue efforts to develop an application called HawkTour – a virtual Illinois Institute of Technology tour guide. HawkTour will provide a completely new approach to the campus tour at IIT. The application will be designed to run on Tablet PC's or other devices with similar computing capabilities, and will provide the user with general campus information while guiding the user around campus and maintaining complete awareness of the user's current location and intent, thereby adapting the tour to the user's own personal preferences. The objective for this semester's team is to build on the framework designed during the Spring 2004 semester, and increase the test bed of the system to include not only location awareness within buildings, but also outdoors. In addition, several new software features will be added, including the ability of the software to interface with hardware devices wirelessly.

Major design goals:

- Understand the current location and provide surrounding campus information. This includes respective building information and history about the building.
- Provide campus information on demand and at all times.
- User-friendly interface in navigation through the campus and in the campus buildings.
- The application should be easily extensible so that new campus tour features can easily be added.

3. TEAM ORGANIZATION

Currently the team is divided into three groups that will each focus on one of the major aspects of the project. Responsibilities are not defined on an individual level; rather, the activities are assigned to a sub-group as a whole. The sub-group is assigned a leader who is responsible for delegating tasks among the sub-group members, and ensuring assigned tasks are accomplished. As the goals for each group are accomplished and the needs of the project change, members may be reassigned to different groups, old groups may dissolve, and new groups may be formed.

- *Application Development Team* – This team is responsible for the actual design and implementation of the actual HawkTour application, including the main application and its subsystems, and both the location and content services. *Members of this team: Robert*

Brozyna, Tyler Butler, Nicu Ilea, Nicholas Lee, Michael Sepcot, Kamil Sobczyk, Dan Wolkenfeld, Yap Yen

- *Design Team* - The design team is responsible for analyzing actual tours to determine useful application functionality, and also designing the application user interface. *Members of this team: Chris Cais, Wonsuk Chung, Andrew Kim*

- *Hardware Interface Team* – The hardware interface team is responsible for developing, implementing, and documenting the HawkTour hardware interface, and also testing it and integrating it into the application. *Members of this team: Aaron Collver, Santhosh Meleppuram, Henry Oyuela*

4. PROGRESS OVERVIEW

The following tasks have been completed by the team this semester:

- Program functional in MTCC
- Design recommendations document produced
- All project documents produced

Application Development Team:
- Entire application rewritten from the ground up
- All application subsystems complete
- Content loading and streaming more effective

Design Team:
- Initial user interface designed and partially implemented
- Analysis of actual tours complete
- Recommendations document produced (See Appendix A)

Hardware Interface Team:
- Design and implementation of interface and API complete
- Integration with application fully complete

5. FUTURE TASKS

The product is nearing completion, but needs to be tested thoroughly. User testing should be done, and the recommendations from the design team should be incorporated.

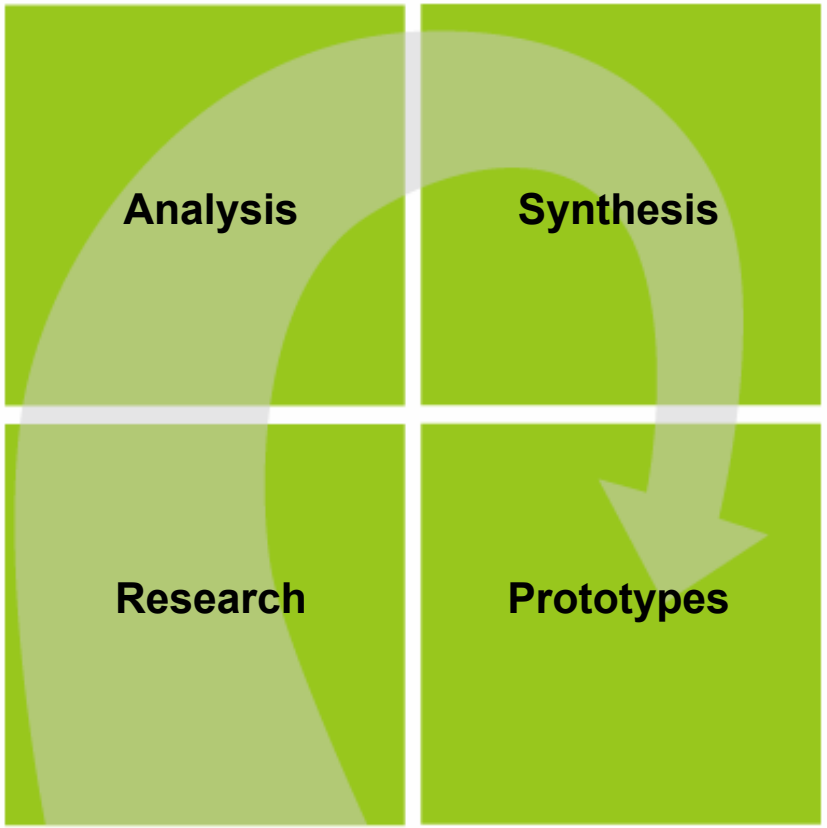
Appendix A: Design Team Analysis and Recommendations

HawkTour:
User Research and Analysis

Prepared by Wonsuk Chung and Andrew Kim
Institute of Design, IIT
November 2004

ILLINOIS INSTITUTE OF TECHNOLOGY

Innovation Process

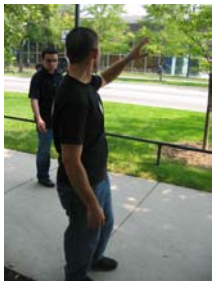


User Research 1

We Observed a Campus Tour… (Sep. 10, 2004)



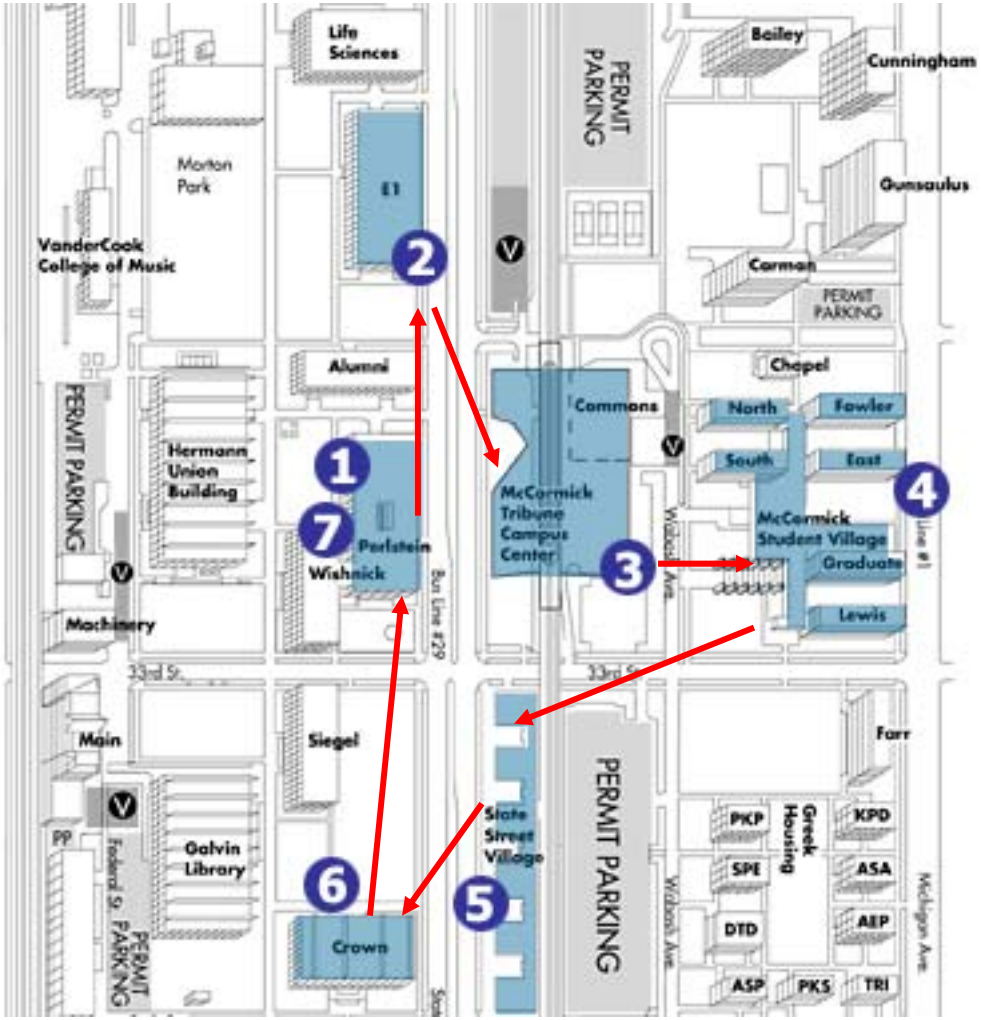





- Friday at noon
- 1 hour tour
- Single student guide
- Group – 3 students
- Appointment suggested
- Offered once or twice a day

User Research 1

Campus Tour Route



1. Admissions office
   - meet guide
2. E1 Building
   - classroom
3. McCormick Campus Center
   - orange walkway
   - wall clock
   - post office
   - grill
   - student offices
4. McCormick Student Village
   - lounge
   - model room
5. State Street Village
   - lounge
   - balcony
6. Crown Hall
7. Return to admissions office

Campus Tour Activity Map

Schedule tour

Contact admissions office → Make appointment

Pre tour

Meet at admissions office → Wait for student guide → Student guide introduction → Participant introductions

Tour

Visit E1 building classroom → Visit McCormick Tribune Campus Center → Visit McCormick Student Village → Visit State Street Village → Visit Crown Hall

Post tour

Return to admissions office → Counseling session

User Research 1

After the Tour, We Interviewed the Student Guide…

Zachary Hartnett
- Second year business major
- Soccer athlete

Comments/suggestions
- Students typically joined by their parents and sometimes their siblings
- Only graduate and transfer students come alone
- A good guide:
  - reacts to the tour group
  - knows what the group wants
- Most people ask questions about:
  - campus safety
  - class load and difficulty
  - tuition

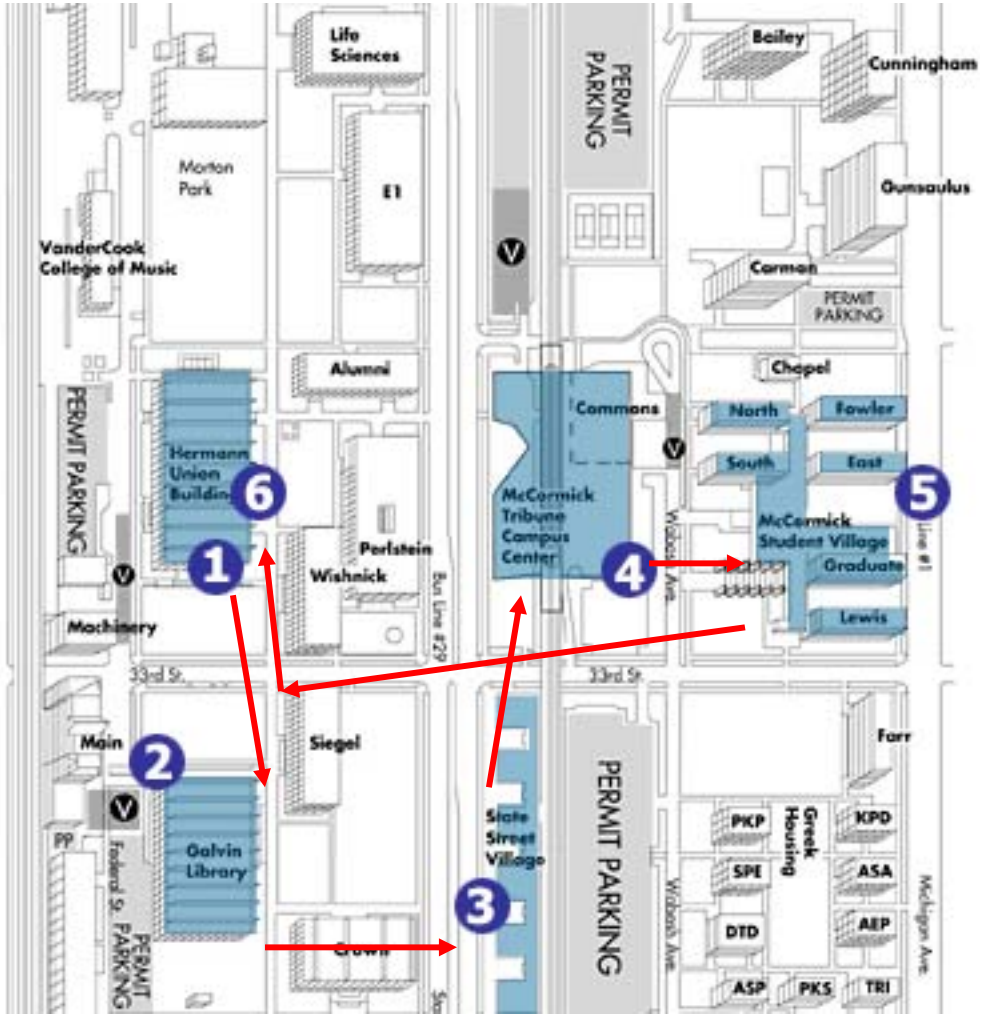
User Research 2

We Observed an Open House Tour… (Oct. 16, 2004)

- Part of weekend open house
- Saturday, multiple times
- 1 hour tour
- Multiple student guides for each group
- Groups of 20 to 30 students and parents

User Research 2

Open House Tour Route



1. Herman Union Building
   - divide into groups
   - meet guides
2. Galvin Library
   - resource center
3. State Street Village
   - lounge
   - model room
4. McCormick Campus Center
   - cafeteria
5. McCormick Student Village
   - model room
6. Return to Herman Union

Open House Tour Activity Map and Era Analysis

Schedule tour

Contact admissions office → Make appointment

Pre tour

Attend morning info sessions → Meet at Herman Union Building → Wait for student guides → Student guides introduction

Tour

Visit Galvin Library → Visit State Street Village → Visit McCormick Tribune Campus Center → Visit McCormick Student Village

Post tour

Return to Herman Union Building → Attend afternoon info sessions

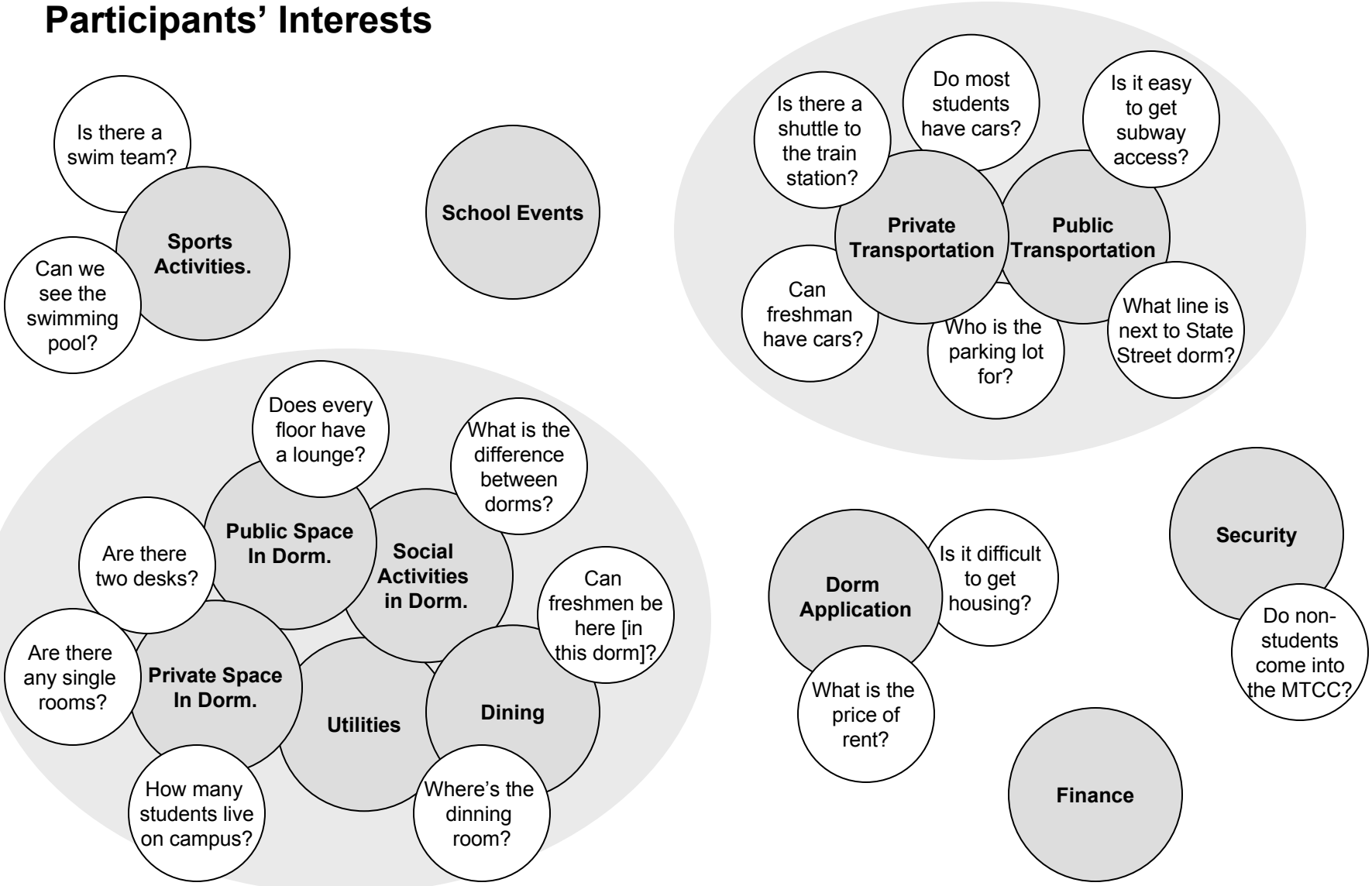
Analysis

We Conducted a Cluster Analysis…

- Collected photos
- Wrote down observations
- Grouped photos and observations into meaningful clusters
- Created group and subgroup headings

Analysis

Participants' Interests



Analysis

Parents ask more questions than students

16 to 2

The number of questions asked by parents versus students on a single tour

Analysis

Key Discoveries Drawn From the Cluster Analysis

1. Tour
   - Rich interaction between student guides and tour group participants
   - More time spent in dormitories than in the student center
   - Limited amount of time spent in McCormick Student Center
2. Guide
   - Student guides customize experience
   - Student guides' use of gestures
3. Participants
   - Prospective students usually accompanied by their parents and sometimes siblings
   - Tour participants asked many questions about:
     - dorm life
     - transportation
     - tuition
   - Parents ask many more questions than prospective students
   - High number of questions asked as the tour group is moving from point to point

Analysis

Insights Drawn From Key Discoveries

1. Tour
   - Successful group interaction starts with knowing participant's interests
2. Guide
   - A positive tour experience depends on the guides' engagement with the participants
3. Participants
   - Tour participants are most interested in day to day campus life and
   - Students are intimated from asking questions in the tour group setting
   - The tour was more parents driven

Synthesis

HawkTour Opportunities

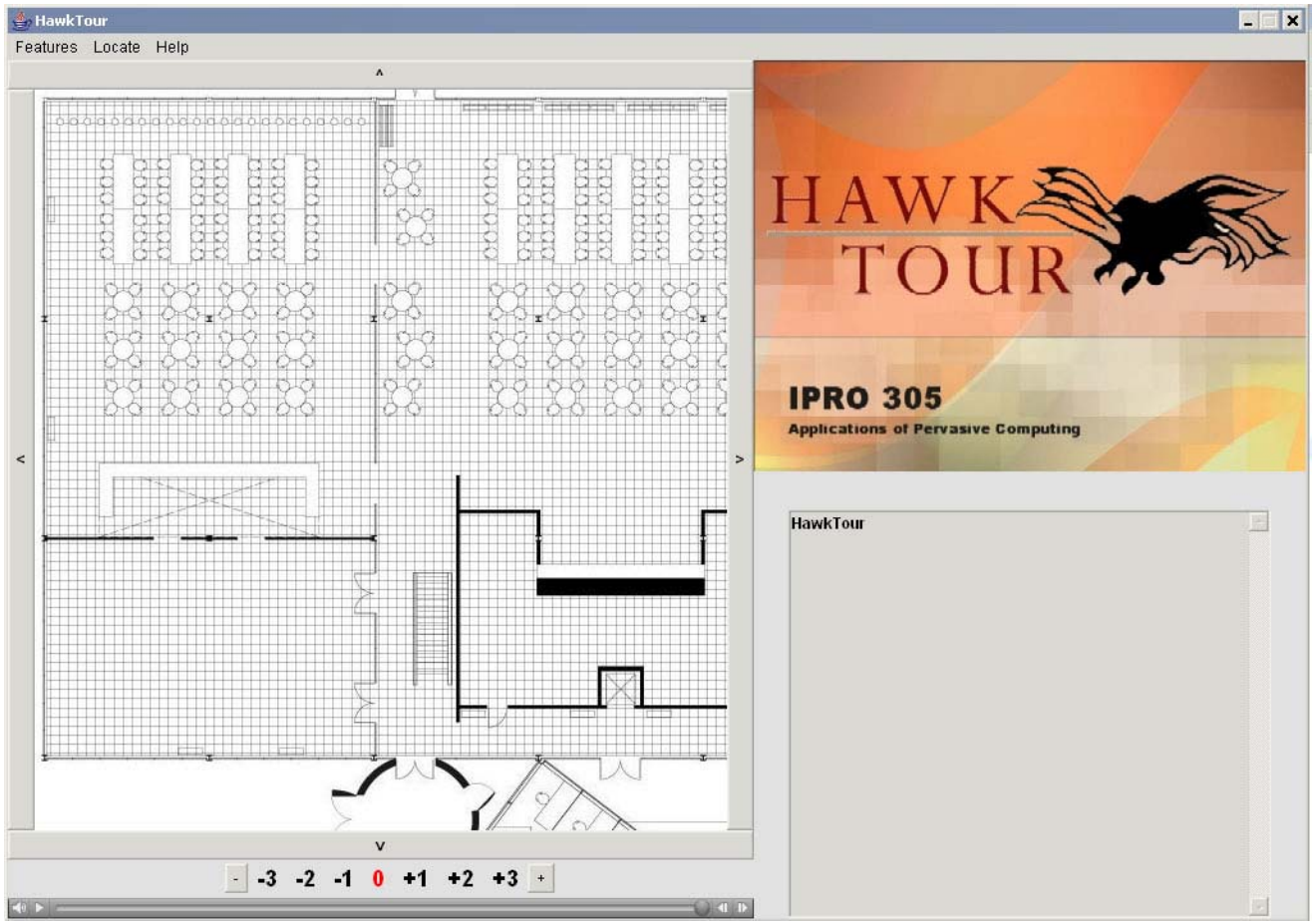
1. Interaction
    - Pre-tour questionnaires
    - Chatting/text messaging with students/admissions office
2. Engagement
    - Participants have the ability to choose the tour topic
3. Contents
    - User generated contextual content
        - Current IIT Students
        - Tour participants
    - Student/Parent Diaries
    - Day to Day Campus Life

Synthesis

HawkTour Open Issues

- Is HawkTour a prospective student or architecture tour application?
- Is HawkTour for prospective and/or currents students?
- Can HawkTour be used for guided led tours?

Old Hawk Tour Interface

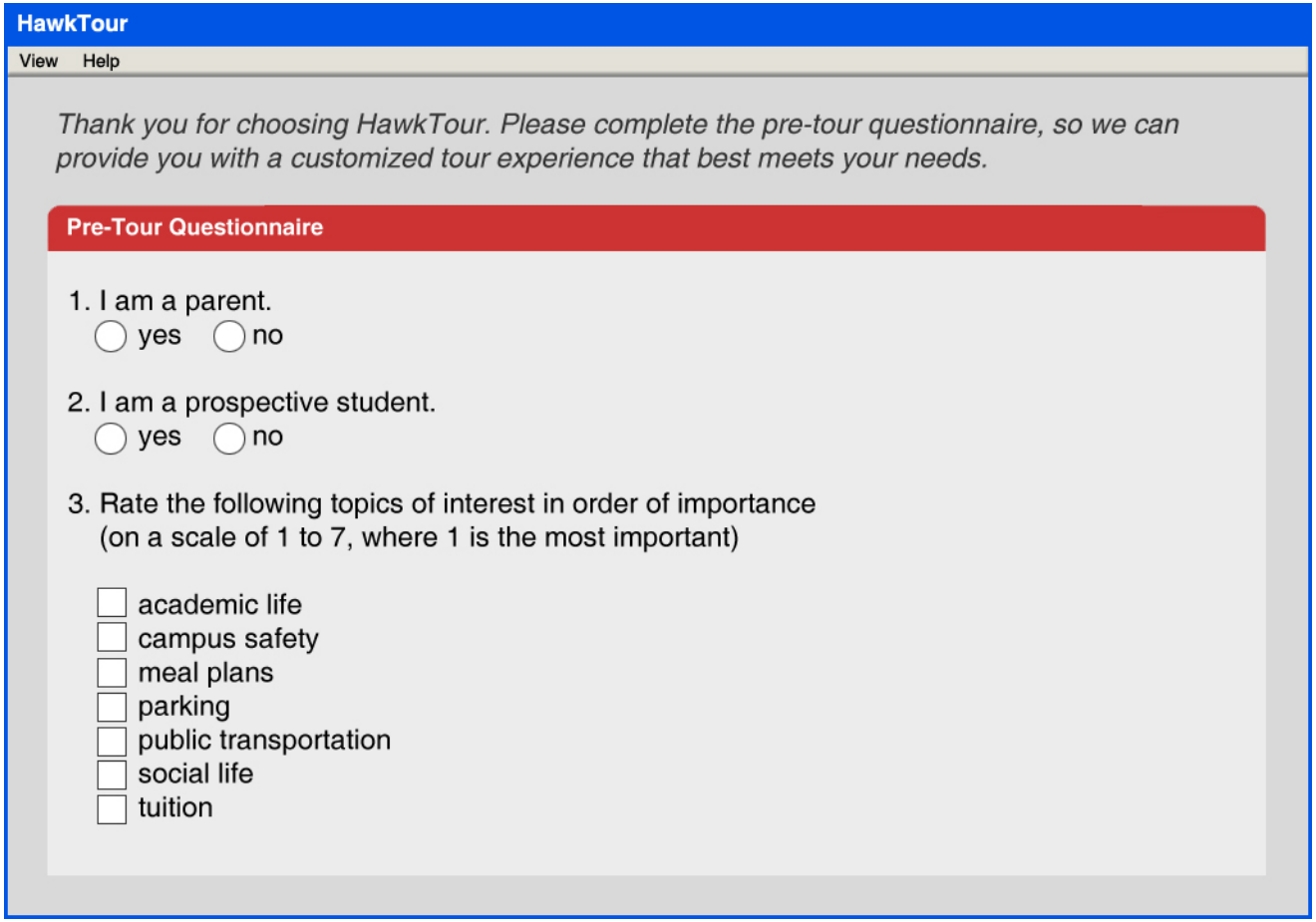


Prototype

Start-up screen options as client application is loading

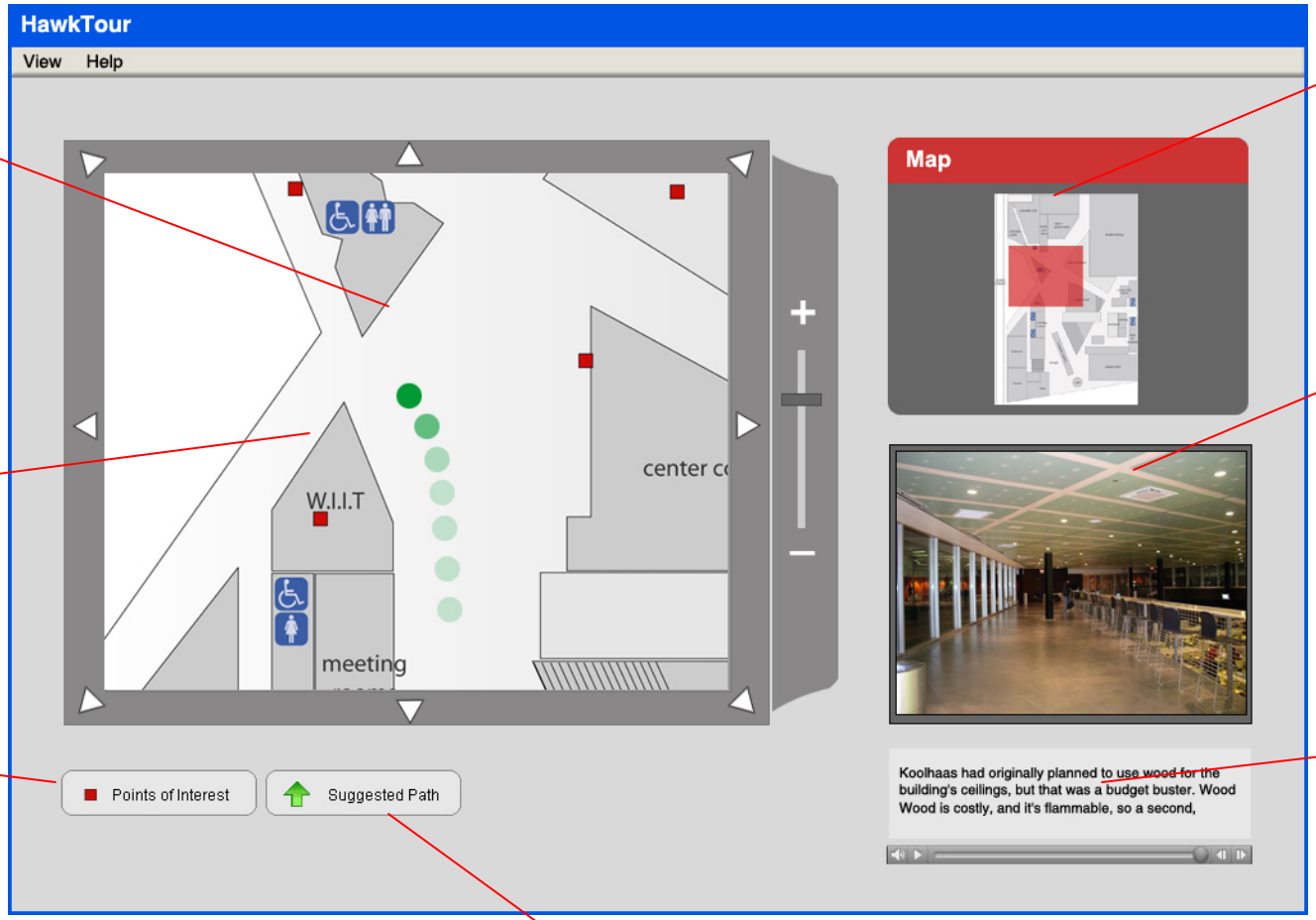
Prototype

Develop a pre-tour questionnaire for tour customization



Prototype

Develop a pre-tour questionnaire for tour customization



Green icon indicates user's location followed by user trail

Point of interest

User can turn on and off points of interest and suggested path

Red box shows user's map screen

360 degree photo

Audio bar

Map provides a suggested path for touring the building, i.e. walking tour

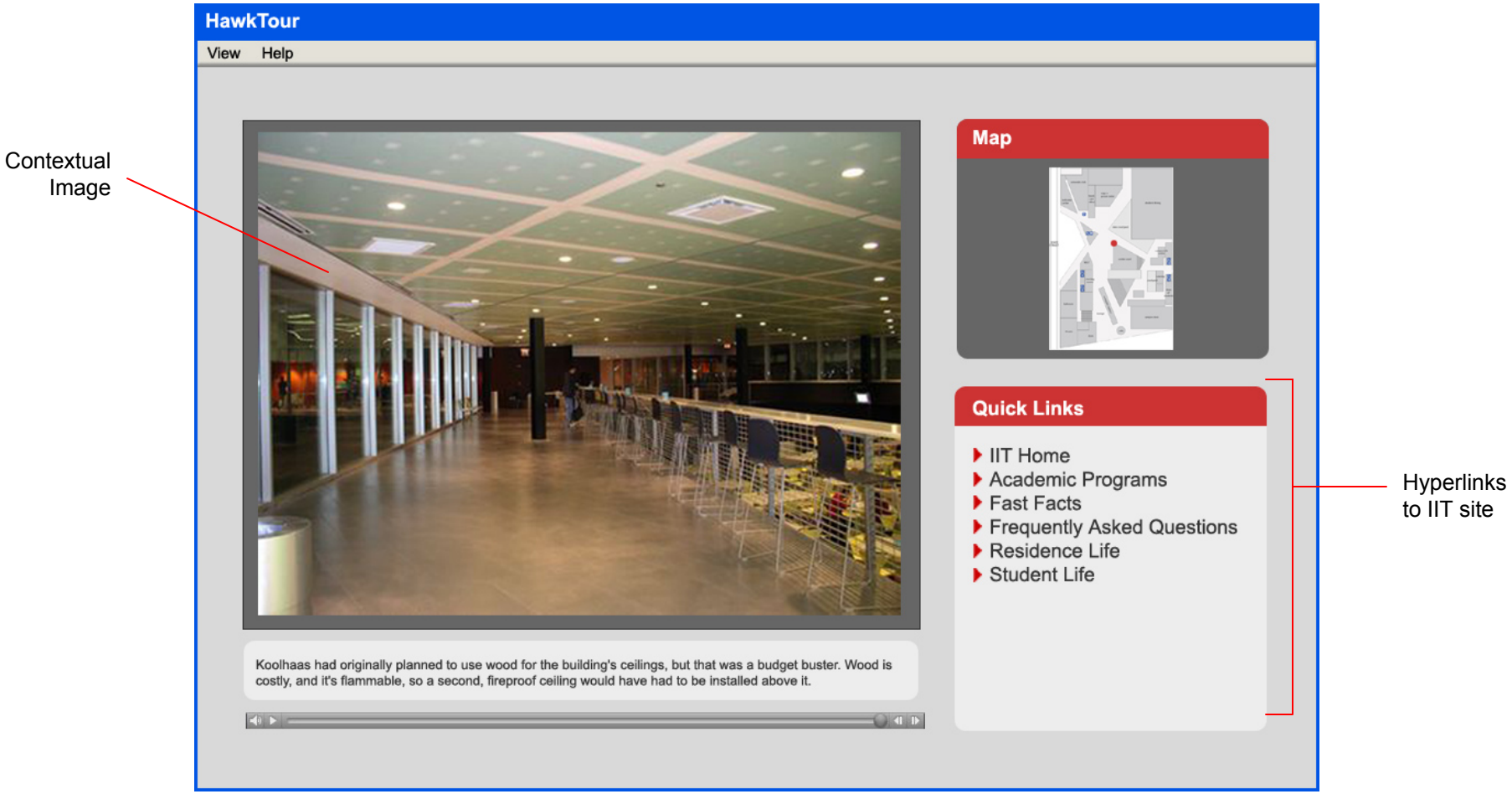
Prototype

Users can view contextual data added by students and/or tour participants

Contextual comments by students

Prototype

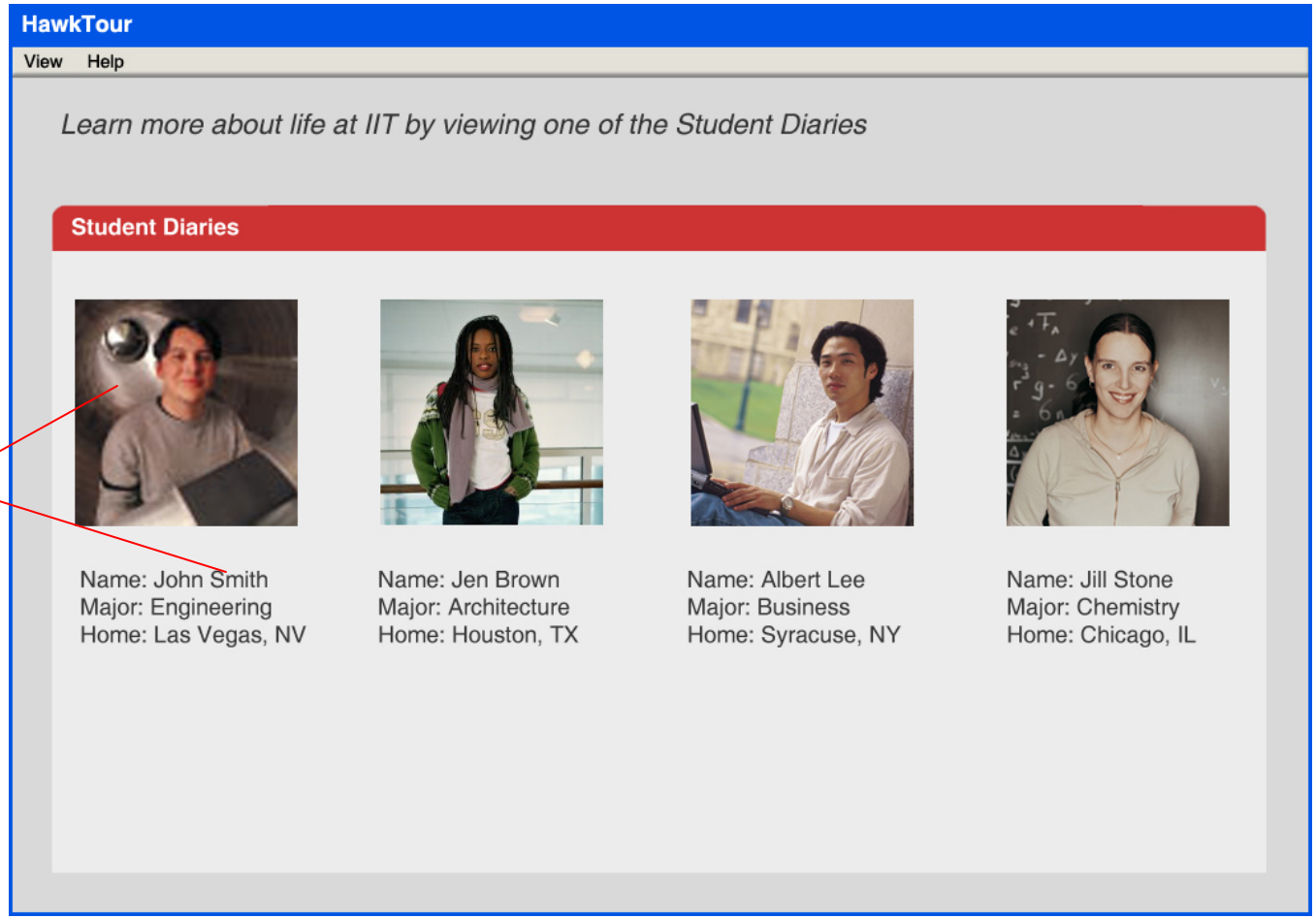
Users have the option of switching to a content view (View → Image View)



Prototype

Users can access student diaries

Click name or photo to view student diary



Prototype

About HawkTour Screen Options

Users can view the About Screen by going to Help → About HawkTour

Appendix

Open House Tour Questions

Parent asked

- Is it easy to get EI access?
- Can we see the school swimming pool?
- Is there a swim team?
- Where's the dining room?
- What line is next to State Street dorm?
- What percentage of students lives on campus?
- Who is the parking lot for?
- Can freshman have cars?
- Do most students have cars?
- Does every floor have a lounge?
- Is it difficult to get housing?
- There are two desks? Is the TV included?
- Can freshman be here [in this dorm]?
- Are there any single rooms?
- Is there a shuttle to the train station?
- Do many public people [non-students] come into the MTCC?

Appendix

Open House Tour Questions Continued

Student asked
- What is the difference between dorms?
- What is the price of rent?

Appendix B: Application Architecture Overview

IPRO 305
HawkTour Developer's Guide
Since 11 November 2004
Daniel Wolkenfeld

Table of Contents

Overview

Scope of This Document

This document serves as a guide for new team members, and a reference for returning team members to the design and implementation of HawkTour. The application has gone through many iterations of redesign and evolution by the successive IPRO 305 teams, but this document is the first attempt to capture the architecture of the functional application as a whole. This document explains how each component and service of HawkTour is designed and why things were implemented the way they were. Undoubtedly, this document will need to be updated at the end of each semester to reflect the new design and features of HawkTour. In the future it would be best to make this document into a web site with links of each class name that occurs linking to its associated JavaDoc page. For more specific details on how the code works, refer to the JavaDocs, and the code itself.

Conventions in This Document

When referring to a class file, as in "class.java" for example, the name of the class will be in blue (RGB: 0,0,128) without the .java extension.

Intro to Pervasive Computing

Why Pervasive Computing? As hardware integration increases to amazing new levels with each generation of design, devices increase in their functionality and versatility. They also become cheaper, smaller, and handier. In tech-savvy societies, people are becoming more and more dependent upon the technology that they carry with them – mobile phones, portable computers, calculators, PDAs, music players, and navigation devices – to the point where an increasing majority of information that people consume and produce daily is in electronic format. With the technology abounding, and with networks connecting them, we can integrate the functionality into a networked application that utilizes this information and makes the technology more useful to us in a whole new way.

What is Pervasive Computing? When a technology becomes widely accepted by society, it tends to disappear. This means that it is there, is easily affordable, and people may use it heavily enough to be dependent upon it, but the technology itself does not require a significant amount of effort or skill to be able to use it so as to get in the way or stand out as an "experience" for the user. A good example of this is electricity. It is available virtually everywhere in the world and people simply plug in whenever they need to, or go equipped with batteries. This has happened with a number of technologies in the 20th century in the Western world, for example transportation, telephone service, and junk food. The user simply benefits from the service of the technology without regard to particular maintenance or ownership of the attendant equipment. The same thing has begun to happen with computer technology. We interact with computers many times on a day-to-day basis, but traditional uses of computer technology, such as controlling the elevator, the traffic light, the ATM, or the telephone auto-attendant at the customer service department, are not *invisible*. In most cases they are a barrier, a substitute for human interaction. Pervasive computing, on the other hand, is *ubiquitous* technology, used as a tool for providing just-in-time information, or to assist users in some way.

Why is this computer system different from all other computer systems? The thing that distinguishes pervasive computing from "a bunch of computers all around us that we use all the time" is that a pervasive application is *context aware*. It takes into account one or more environmental data such as your location, the weather, the time, your personal information, your biometrics, etc., in determining its behavior. A pervasive application also uses shared data from other systems, applications, or services.

Web Services and HawkTour

To facilitate the needs of a mobile, pervasive application, it is very convenient to transfer data across components of the application using Web Services. This is not only convenient, but it allows HawkTour modules to be virtually hot swappable, so that the application does not have to rely on any particular technology. In terms of modules, right now the major components of HawkTour are the Location Service, the Content Service and the local HawkTour client, of which the Location Service and Content Service are web services. As a side-note, a middleware web service called Scarlet can be used to broker these and other web services with HawkTour. However, Scarlet is not important to this version of HawkTour (for more information about Scarlet, talk to Tyler Butler). So in the future, this application may interact with more web services for retrieving other contextual data.

Model View Controller Paradigm

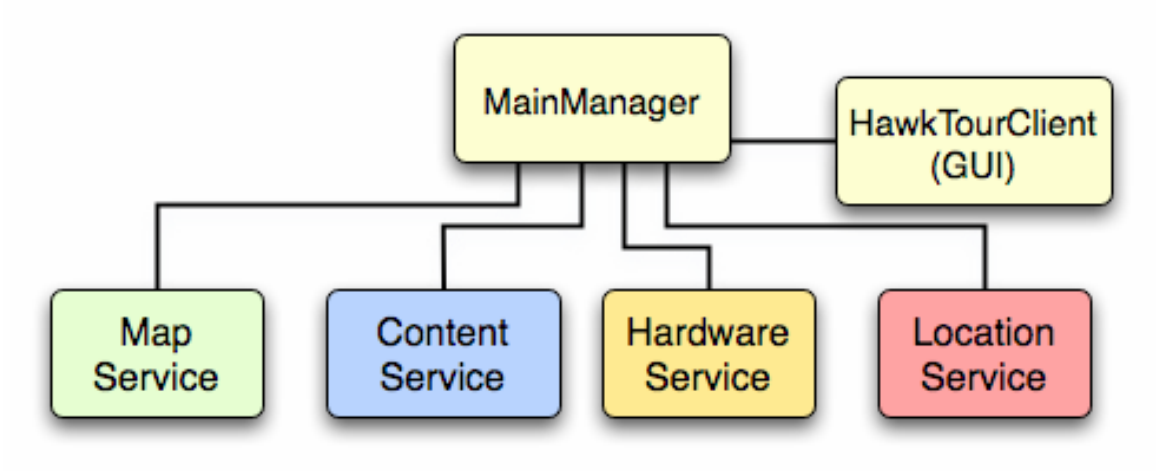
Referred to as MVC, this software architecture concept was first developed at the Xerox Palo Alto Research Center for their Smalltalk-80 system for the multi-windowed graphical interface – the first in the world. Succinctly, the Model section of the program is the function, algorithm, or state-machine; the View is the output, usually a rectangular portion of the screen; and the Controller is the input handler / interpreter. Within this triad there is heavy communication between the View and the Controller, and communication between the model and each of the view and the controller. HawkTour is built on the MVC design pattern, with the HawkTourClient (GUI) corresponding to the View, the MainManager corresponding to the Controller, and each of the back end services corresponding to the Model.

Application Architecture

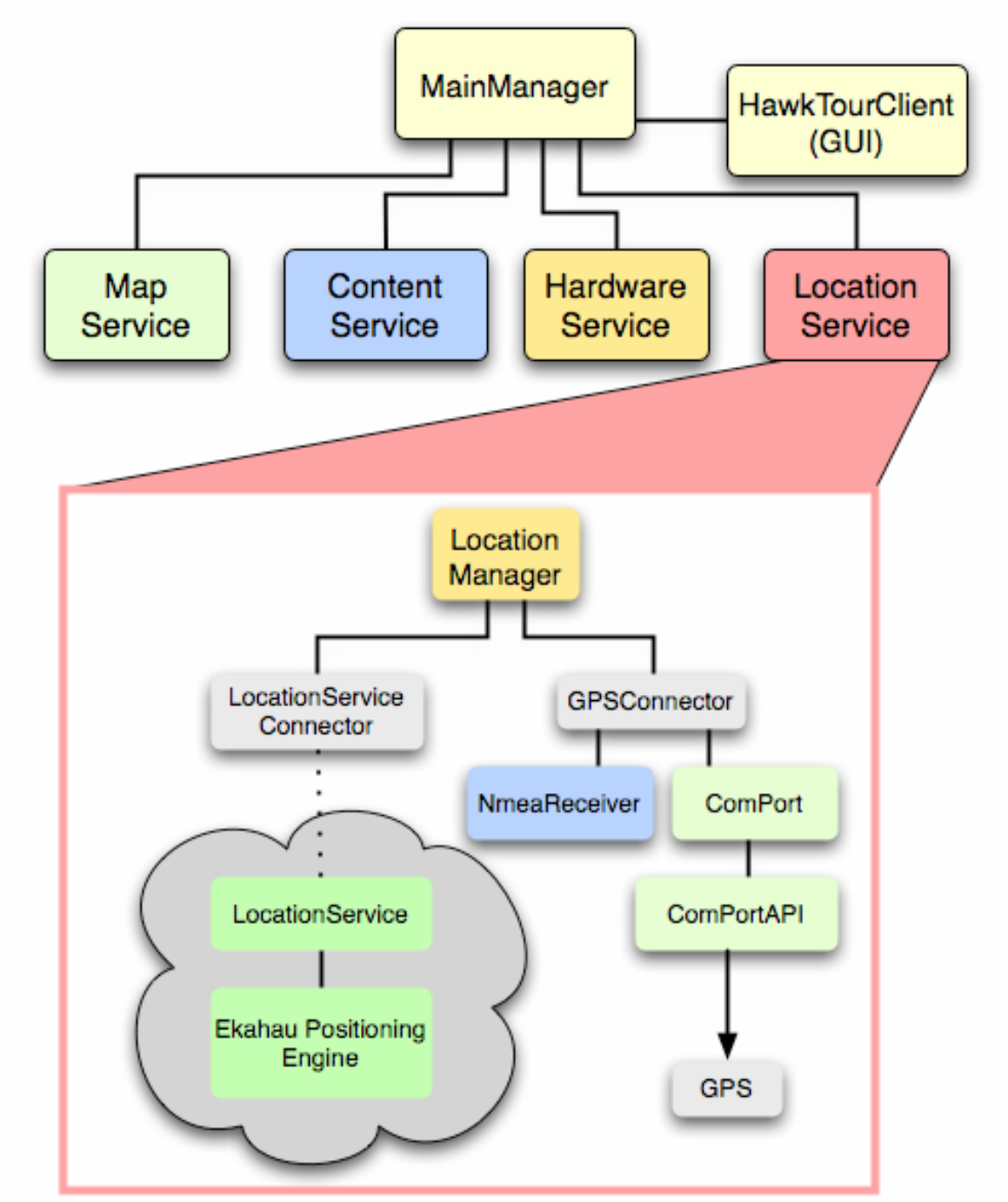
Application Hierarchy

HawkTour went through a design overhaul over the summer of 2004 to make it more organized and hierarchical, and to prevent headaches from reading the code. The main purpose, besides code readability, however, was to implement a consistent interface for the application to communicate with the different services. The motivation for this was to make the different services completely independent of each other, so that the application would not be inherently tied to a technology used to implement one of the services. For example, right now Quicktime is used for content delivery, and Ekahau is used for WiFi positioning. Either of these components can and may be upgraded to newer or more favorable technologies in the future, or another development group may want to substitute a technology more appropriate for their purpose for their version of the ____Tour application. The graphs on the following pages illustrate how each of the component services works.

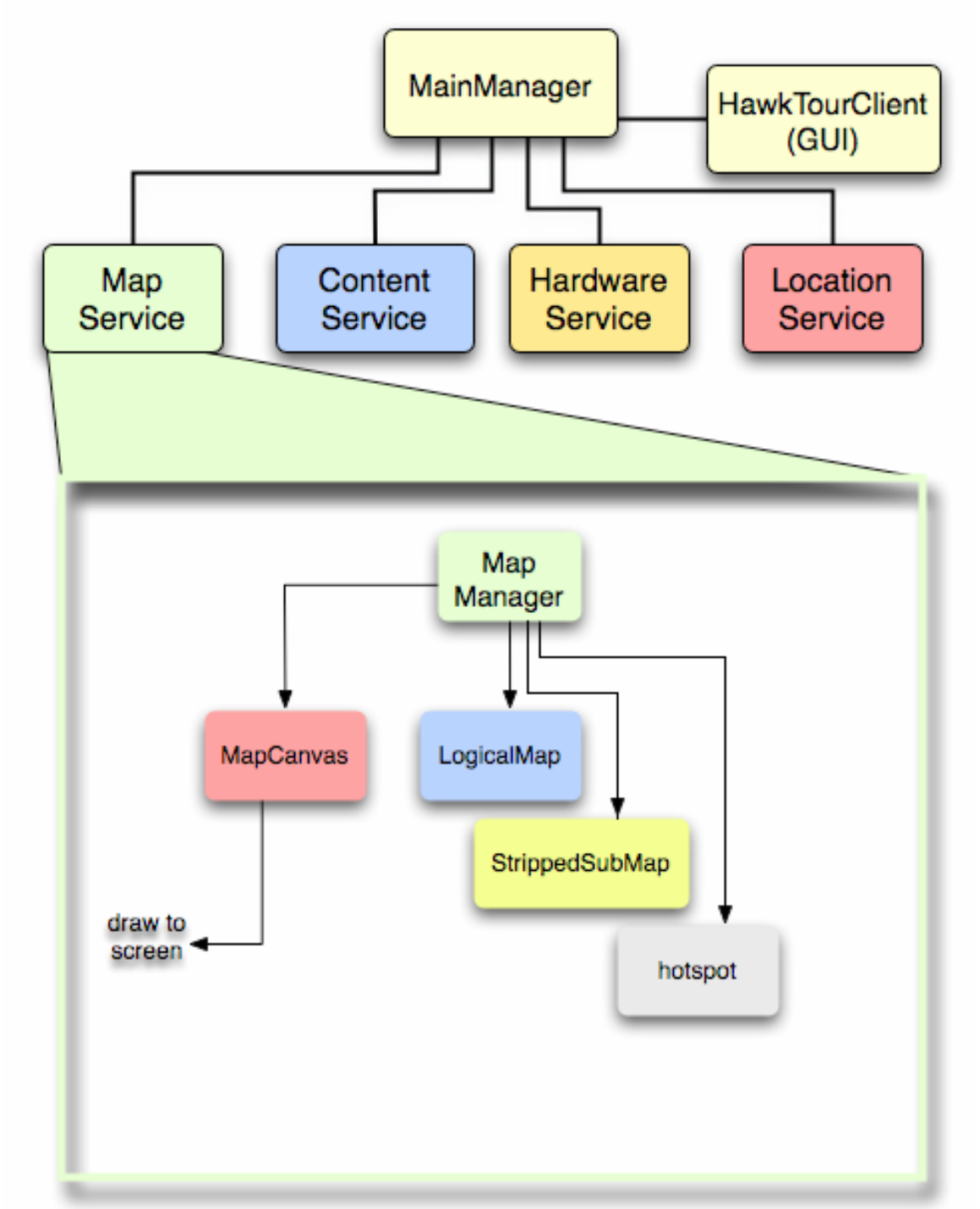
HawkTour Hierarchy



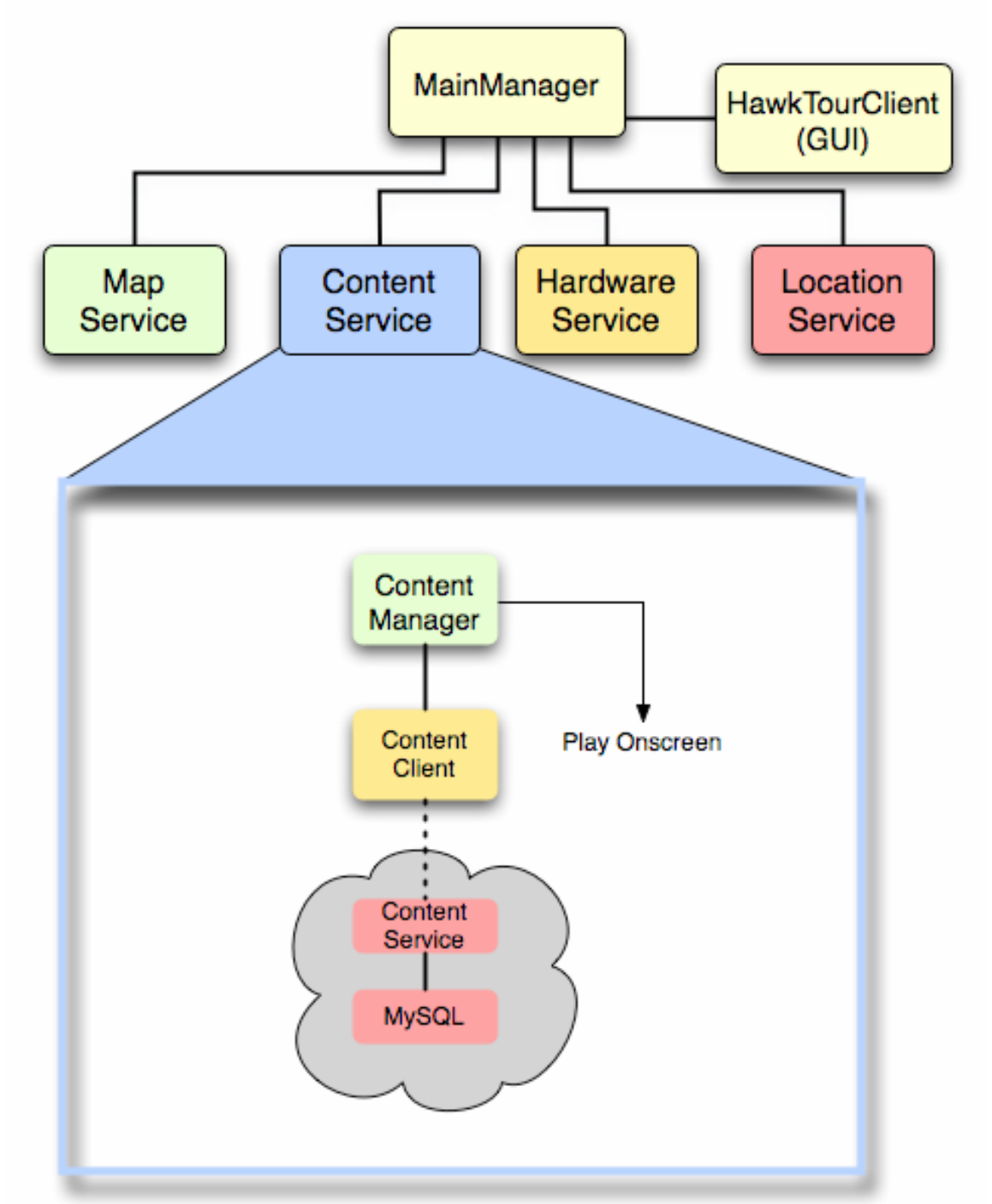
Location Service



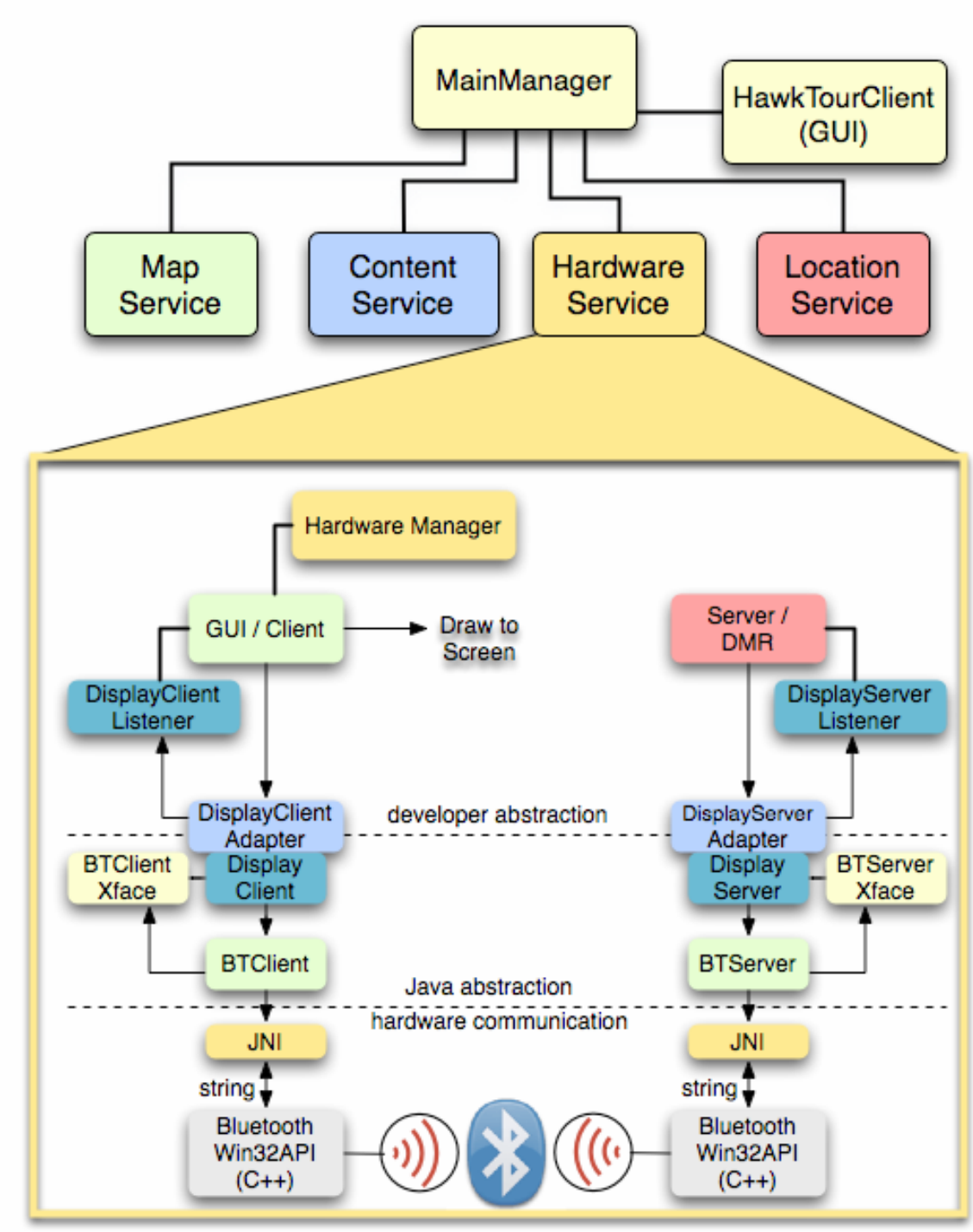
Mapping Service



Content Service



Hardware Service



Future Work

It is of token value to mention here that a nice extension of the application in the future would be an update service, which could also be implemented as a web service.

Application Implementation

MainManager

MainManager is the mother of all classes of HawkTour. This class was written in Fall 2004 as part of the move to de-spaghettify the code and make it more hierarchical. It is a Singleton class, and creates instances of the "middle managers" {LocationManager, MapManager, ContentManager, and HawkTourClient}. It acts as the middleman between them. If, for example, the application needs to find the current location, you just call the name of the MainManager (mainMan for the sake of argument), as in mainMan.Location().

LocationManager

The LocationManager class is another result of the move towards de-spaghettification. It is in charge of gathering location data from the LocationService. The LocationService, in turn, connects to the locator servers and puts the information in a plain old Vector, called LocationData.

The format of LocationData is (in Global coordinates) X (longitude), Y(latitude), speed, heading.

There are currently two location services: WiFi and GPS. The LocationServiceConnector (package edu.iit.cs.scs.HawkTour.ServiceCommunication) retrieves the location data from the WiFi location server, which right now uses the Ekahau Positioning Engine. If a different technology becomes more favorable in the future, a separate Connector class would be written for it, and the LocationServiceConnector would be deprecated. GPSConnector, of course, gets the information from the GPS device, which "happens to be" on the local host, through the ComPortAPI class (package edu.iit.cs.scs.HawkTour.LocalServices). But it is cleaner to have a separate Connector for each locator service, and they all return values to LocationService.

In addition to gathering data, LocationManager also arbitrates which locator service the application is using, so that the source of information is hidden from the application. It simply asks LocationManager for the current location, and is returned an (hopefully) accurate value, from whichever service the LocationManager deems most suitable for the present conditions. Currently, automatic arbitration is not workable, so a manual mode switcher is being used.

MapManager

MapManager controls the classes that are related to the mapping service in HawkTour. Right now we only have one MapManager constructor and it accepts a MainManager object, 2 floats, 1 string, and a dimension object as its starting values. Using the setLocation function, which will be called from the MainManager every 10 seconds or so (a fixed time), the current PixelPoint and currentGpsPoint is set together with the currentPixelView, which will be later sent to MapCanvas to draw. The gpsToPixel class is used to convert GPS coordinates which are obtained to Pixel

coordinates, which is used to display graphics on the screen. The MapManager keeps all the submaps for the current location in a vector called mapStack. When the current location changes, MapManager uses is_In functions to determine whether the current map is still valid. If it is not, and an adjacent map needs to be displayed, it traverses up the stack until the is_In function returns true, then uses the findSubmap function to find the smallest available map for the current location.

MapCanvas handles the display of the map. MapCanvas takes in an image of a map, draws a portion of it and displays it as our current viewable map. It also handles the size of the user location dot.

LogicalMap's main function is to take in a map file, parse it, and load up the appropriate Submap (*.sub) and Hotspot (*.hot) files. The HotSpot class' job is to just return the hotspot ID and the coordinates of the hotspot to LogicalMap. The StrippedSubMap class is similar to the HotSpot class, but instead returns the coordinates of the Submap and the name of the Submap found within the coordinates of the current map. The coordinates mentioned are all global (GPS). In addition, LogicalMap provides is_In and findSubMap functions for MapManager to call when searching for the smallest available map to display.

ContentManager

ContentManager receives a list of URLs from ContentClient, which could be better named "ContentServiceConnector." ContentClient uses SOAP to communicate with the ContentService. ContentService is basically a Java database accessor with a bunch of getter functions that are called by the getter functions of ContentClient.

HawkTourClient

The HawkTourClient handles the rendering of the main window, the user interface which we see when we run the application. It makes calls to MainManager and MapCanvas. It generates the map window (zooming/scrolling buttons and main map display), and menu options (buttons).

Hardware Service

The hardware section of HawkTour is new as of this semester (Fall 2004), and so far it supports Bluetooth functionality. With Bluetooth, the application is able to communicate with DMRs (Digital Media Receiver). A DMR is a Bluetooth enabled PC whose video output goes to a large-screen TV or HDTV and digital media is stored on it. The DMR is called upon by the Bluetooth client (i.e. HawkTour running on a tablet PC) to start displaying pictures or video. This is done when both devices, the tablet and the DMR, discover each other through Bluetooth when in near proximity.

In the HardwareService diagram above, the Bluetooth functionality is divided in two: the client and the server side. On the client side, the GUI / Client is implemented in a "SampleClient" (a generic implementation written by a generic developer who wishes to use the Bluetooth function), which extends the DisplayClientAdapter. This setup allows the developer to implement just a subset of the methods in DisplayClient.
The SampleClient receives events to display to the screen through the DisplayClientListener, which defines the events that are received from the DisplayClient, and sends messages over to the DMR through the DisplayClientAdapter. This way,

SampleClient and DisplayClientAdapter do not need to make constructors for each other, and message passing is simpler. The user (generic developer) never needs to know what is behind the DisplayClient. Similarly, the DisplayClient receives messages through the BTClientXface from BTClient, and sends messages to BTClient. The BTClient, responsible for device detection, device info, and device connection parameters, is the lowest level that the application transmits the messages before they actually enter the Bluetooth hardware. The Bluetooth Tx / Rx hardware is dealt with by the Java Native Interface and the Win32 API.

On the right side is the DisplayServer class, which is called by SampleServer, or any other class that needs to use it, and DisplayServer will return data from the BTServer class. Again, the user will never need to know what is behind the DisplayServer class when writing an application like SampleServer. DisplayServer provides the functionality needed to create a server by encapsulating Bluetooth functionality and message passing. DisplayServer provides events through the interface DisplayServerListener. DisplayServerListener defines the events that are received from the DisplayServer.