SmartCIT

An Intelligent Sensor Network System

Vince Liang (Vince_Liang@brown.edu)
May 16, 2006

***Abstract* – The SmartCIT project is an attempt to merge information from disparate data streams through a custom query engine to answer interesting and complex queries about the environment and the people in it. The information for these data streams is collected through remote sensors (some wireless, some wired) strategically dispersed throughout the test space. Currently tested data streams includes personal location information, provided by a modified implementation of Harvard's MoteTrack system, and image processing modules, which analyze data received from web cams. Major functionality of the system includes monitoring coffee pot level, protecting projectors idling overtime, head counting in a room, and maintaining room schedule. Database connectivity and web service have also been deployed in the project. The current deployment of this system stands as an interesting proof of concept, and it could be extended in any number of ways.**

Key works: SmartCIT, Sensor Networks, MoteTrack, Automatic System, Monitoring System.

I. INTRODUCTION

SmartCIT is a project idea developed in the Computer Science Department at Brown University. It was originally initiated as a course project for a team of four

players in CS295-1 (Sensor Networks) in the fall semester of 2005. In the spirit of that course, the primary goals of this project have been to identify and explore the limitations of current sensor network technology, while creating a novel system that combines data from a variety of sensor types to serve to monitor the status of the building (Center for Information Technology). That is how the project name SmartCIT comes. After the semester was over, the project has been maintained, modified, and further developed by Vince Liang. The current incarnation of SmartCIT uses a database engine, a modified version of the MoteTrack location tracking system, several web cams, and a web interface to gather, process and display information about the environment, as well as the results of an interesting set of queries.

It currently stands as a proof of concept system deployed over a small area of the 3rd floor atrium of the Center for Information Technology at Brown. The area of interest includes the atrium, the adjoining kitchen space, and the nearby meeting room. As an overview, the system features some interesting functionality. Here are some examples. The system knows the number of people in a room. With the maintained room schedules in the database, it is easy for a professor to find an empty room for a group meeting without going around the whole building. During the meeting, people take a break and brew coffee in the kitchen. But it takes a while and they do not want to wait. So they go back to the meeting with confidence that the system waits for the coffee. When the coffee is ready, the system automatically sends text messages to the registered cell phones, informing those people that coffee is ready. After the meeting, people leave the room without turning off the expensive projector. No problem. The system keeps an eye on it. When the smart system finds no one in the room and the projector keeps on for a long while, it automatically alerts the nearby staff for immediate attention. This illuminates the way the systems works.

The rest of the paper is structured as follows: section 2 will describe the main structure of SmartCIT's implementation. Section 3 will explain how the web cam related programs of SmartCIT work. Section 4 will discuss the use of MoteTrack location tracking system and its unsatisfactory performance. Section 5 will discuss the database

engine, which acts as a centralized data warehouse for the whole system and bridges the communication of various components. Section 6 will reveal the function of automatic notification. Section 7 will briefly explain the design behind the web interface. A conclusion will follow.

II. SYSTEM STRUCTURE

Currently, the SmartCIT system includes several components: a web cam driver with motion detection capability, image processing units, a database engine, an event notification module, and a web-based user interface. Web cams function as sensor nodes of the network, which, under automatic control by the driver, collect various kinds of status information from interesting spots. The driver also detects motion in the captured images and thus tracks the number of people in the room and feed this information to the database engine. The image processing modules are designed to perform various tasks, ranging from watching coffee pot level in the kitchen to monitoring the projector status in a lecture room. The modules perform such tasks by processing the image streams provided by the web cam driver. Just like the driver, thse modules also communicate with the database to retrieve and store information. Once a pre-registered event (e.g. coffee pot is full and ready to serve or projector has been on for too long but nobody is in the room) happens, these modules call an external notification program (event-notify) to send an alert message to a registered cell phone. The database stores information received from the various sensors and provides useful status information to both running programs and end users. Normal users of the system access the data with a web-based interface. Using a web browser, the user sends requests to the web server where php programs queries the database and return to the user in html format.

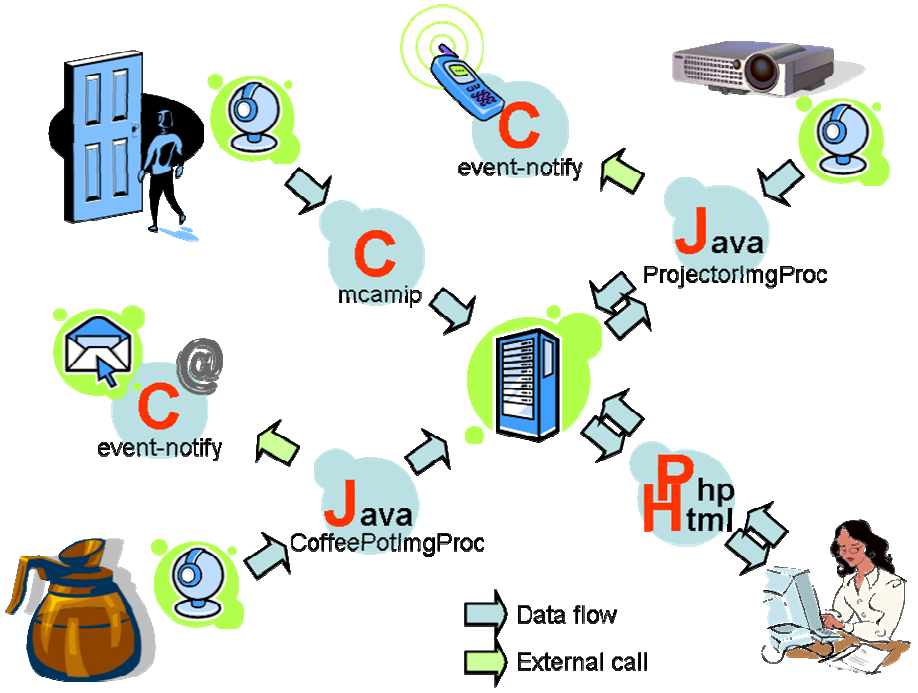


Figure 1: An illustration of how various components of the SmartCIT system work together.

Previously, a wireless location tracking system developed by Harvard University, MoteTrack, was also included in the system. With motes (Berkeley Mica2/Mica2dots) functioning as distributed beacon signal generators, MoteTrack receives data from motes and processes the data to calculate the location of the object being tracked. Even though a working MoteTrack together with other sensors would provide a lot of interesting queries to the SmartCIT system, we had to discontinue working on MoteTrack due to its poor accuracy proved in the early stage of the project.

III. Web Cam Related Programs

In this project, we utilize web cameras to provide real time image or stream data for complicated and interesting queries. To get the camera data, we use a modified third-

party program to communicate with the camera. Customized java programs are employed to monitor coffee pot level and projector status. All these programs are capable of communicating with the mysql database.

1. Camera Driver (mcamip.c/video.c)

The web cameras that we have utilized in this project are D-link DCS-900w. This web camera provides two types of connections: wired ethernet and WiFi. A software package called mcamip was developed under the GPL license to drive the camera in linux platforms running X windows. In order to make the software work with our specific project, the programs have been modified and some additional features have been added.

The program is written in C and works directly with the camera and the mysql database with the support of necessary libraries. It connects to the camera via the HTTP protocol.

When the program is initiated, a socket connection is set up with the webcam via port 81 of the webcam's IP address. An HTTP request is then sent to the webcam with necessary information as follows.

The request looks like this:

```
--------
GET /video.cgi HTTP/1.1
User-Agent: mcamip (rv:VERSION; X11; Linux)
Accept:text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.
8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,text/cs s,*/*;q=0.1
Accept-Language: en-us, en;q=0.50
Accept-Encoding: gzip, deflate,compress;q=0.9
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Keep-Alive: 300
Connection: Keep-Alive
```

Authorization: Basic USERNAME_PASSWD

Referer: http://SERVER_IP_ADDRESS:PORT/Jview.htm

After the connection is set up, the program starts reading from the socket. What the webcam sends in is a stream of jpeg raw data. The program detects every jpeg frame in the stream and save the jpeg file into a specific folder. As a new jpeg file becomes available, the program overwrites a previously saved file. This practice ensures the file in the specific folder is always up to date.

Once the program is initiated, it keeps producing the image file until it is manually stopped.

One of the major modifications is the motion detection capability. This modification is so designed that it tells the direction that a mobile object moves. The room head functionality of SmartCIT is heavily based on this motion detection unit. When the module analyzes the camera images, every frame is evenly divided into several horizontal areas. Pixel values are then compared with previous recorded values in order to detect possible pixel in these sub-areas. Any motion detected in these smaller areas are recorded by the program, which keeps a history of the detected motion of the same mobile object and thus determines the direction in which the object moves. It is inevitable that noise exists in the camera images which could result in false detection of motion and wrong moving direction. With the influence of noise taken into account, the program employs some algorithms to reduce the probability of false detection. The program examines all sub-areas for pixel change and eliminates all those that could lead to inconsistent history of the moving object. For an example, if most sub-areas where motion is detected are the leftmost ones, then an area that is also marked “motion detected” could be just noise caused by other small interference and could be safely discarded. Based on the motion detection, the program keeps track of the number of people who enter or leave the room and thus increases or decreases the head count. Once the head count is changed, it immediately updates the database.

Generally, the web cam driver works fine and is capable of counting the number of people in the room if properly set up. Since the lighting and background setting vary from room to room, this motion detection based algorithm does not work very well when program parameters are not set correctly. In order to ensure the system's accuracy and reliability, however, careful setup of the camera and fine adjustment of the program parameters are required.

2. Coffee Pot Level Monitoring

To prove the concept, we designate a D-link DCS-900s web camera to monitor the coffee pot on the 3rd floor in the CIT building. The camera is mounted close to the coffee pot in the kitchen. To perform the pot level monitoring, both the camera driver and the image processing unit need to be started. Once initiated the camera driver outputs real-time coffee pot images to a specified directory and updates it in a reasonable short period. It only provides raw image frames without any processing. On the other hand, the monitoring program (CoffeePotImgProc.java) reads and analyzes the images. The program runs in an X window in which an up-to-date coffee pot picture is shown as well as the status information.

To implement this functionality, a rectangle monitoring area has to be defined before the monitoring program is able to determine the pot level. We define the bottom of the area as coffee pot empty and the top of the area as coffee pot full. This can be done by using the left mouse button to set the lower left and upper right corner of the area. We have a picture showing the X window after defining the rectangle monitoring area (Figure 2). A reset function can be called by mouse right click if necessary. Once the monitoring area is defined, the program starts to analyze the current status of the monitoring area, displays the percentage of fullness in the X window, and updates the database server when the pot level changes.

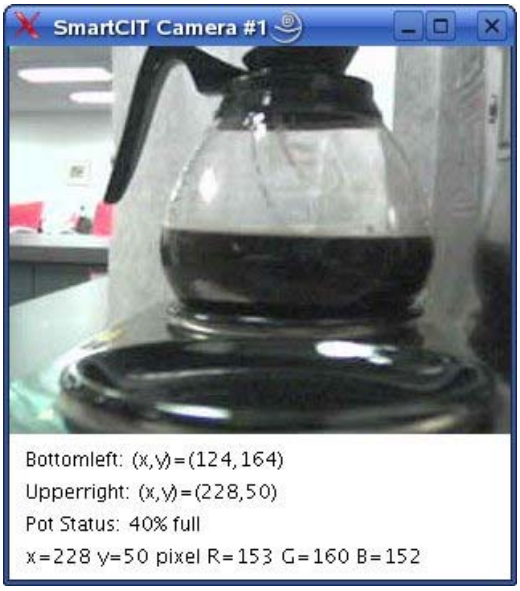


Figure 2: A. A real-time image showing the X window produced by the java program CoffeePotImgProc.java.

The CoffeePotImgProc class uses an underlying ImgPixelGrabber class to get the RGB values of critical pixels of the coffee pot frames produced by the mcamip program. The RGB values are used to determine the dark/bright state of a pixel. The monitoring area is viewed as a number of narrow horizontal stripes. Each of the stripes has it own status of whether coffee reaches this stripe. We determine this status by analyzing the color information of a number of evenly distributed pixels in the stripe. If a majority of pixels in the same horizontal level are determined dark, we deem that coffee reaches this level.

Besides outputting the coffee pot status to the X window, the program also updates the corresponding record in the database containing the status information using the connector/j, a native Java driver that converts JDBC (Java Database Connectivity) calls into the network protocol used by the mysql database. A real time image stream is open to the public through the web service.

In case a full pot is detected, the email notification program is called to send an email or SMS message to a pre-registered device, informing people that coffee is ready to serve.

3. Projector Status Monitoring

Nowadays many multimedia classrooms are equipped with overhead projectors for slides showing or video demonstration. A typical projector lamp is a very costly consumable and it has a limited lifetime up to 1000 ~ 2000 hours. Heavily used projectors need to replace bulbs often. SmartCIT aims to increase bulb lifes by actively prevent unnecessary projector usage. When no one is detected in the room and the projected has been detected to be on for a period, the monitoring program alerts the pre-registered person.

The monitoring program is ProjectorImgProc.java which shares the ImgPixelGrabber class with CoffeePotImgProc.java. Starting the service is very similar to that of the coffee pot monitoring module in that the camera driver and the monitoring program need to be both started.

The functioning camera is mounted somewhere close to the projector or the screen. Like the coffee pot case, the drive keeps updating the projector image stored in a specific directory and the monitoring program reads and processes the frame. Unlike the coffee pot case, ProjectorImgProc has an internal timer for the alerting services and it needs to periodically query the database. The program keeps track of the number of people in the room by asking the database every few seconds. Whenever nobody is detected in the room (head count = 0) and the projector is on, the timer is set. The timer will go off and an alert will be sent unless someone is detected to enter the room in which case the timer will turned off.

Technically similar to the coffee pot monitoring scenario, ProjectorImgProc also utilizes image processing units to do the work. When a majority of pixels in the

designated area are determined to be highlighted, the projector is considered on. Multithreading is used to query the database engine and setting the timer.

IV. MoteTrack

MoteTrack is a location information service which was originally developed at Harvard University and uses wireless motes (Mica2 and Mica2dot) distributed by UC Berkeley. The MoteTrack version used in SmartCIT is version 2.0. While the original documentation of MoteTrack version 2.0 claims that the 2.0 system supports Mica2 and Mica2dot which are the only motes available to us, it eventually turned out that version 2.0 could not accurately estimate the location with Mica2 and Mica2dot.

The primary problem we have faced in using MoteTrack is that MoteTrack version 2.0 is primarily designed for the MicaZ and Telos sensors instead of those available to us. From the documentation, MoteTrack version 2.0 is supposed to support the tracking for Mica2 and Mica2dot. However, there are many hidden bugs that were missed when the original author of MoteTrack ported the application to work with MicaZ and Telos. To make the MoteTrack version 2.0 work with Mica2 and Mica2dot, some of the old functionalities of version 1.1 need to be ported in. It may seem as though version 1.1 should have been used, if it is definitely compatible with the hardware, but there is a problem with that. The necessary Java code that generates the database for MODE_NORMAL is omitted in version 1.1. Also, the header files, which define the message types, would need to be changed. We were too deep in our own project to make such a drastic change.

Conversations with original author of MoteTrack have got us some beneficial advice on how to fix the hidden bugs inside MoteTrack version 2.0 to make it more compatible with Mica2 and Mica2dot motes. Although the attempt to trim the MoteTrack version 2.0 to suit Mica2 and Mica2dot is quite successful, the modified version of MoteTrack does not always estimate the current location of the mobile node accurately.

Sometimes the error of the distance between actual and estimated locations can be more than 5 meters. In most cases, however, MoteTrack does err less than four meters which is what the original author proposed as the optimal case. If mobile node stays still for more than three seconds, the estimation usually comes very close to the actual location. There may be several possible reasons for the insufficient accuracy found with MoteTrack. First, unlike the MoteLab at Harvard University, the sensors used in the CIT building are not provided with a permanent power source. Based on the hardware specification, the radio strength from Mica2dot can vary if the power from the battery is less than 3 V. Also, possible location changes of furniture or other obstructions can affect the radio signal strength.

Due to its poor performance, we have decided not to include the MoteTrack in the SmartCIT system.

V. DATABASE SERVER

In order to coordinate the communication of various components of SmartCIT, mysql database is utilized as a centralized data warehouse. Since the functional modules of SmartCIT are written in different languages such as Java, C, and PHP, all of them need to be able to access the mysql server.

A database named “smartcitdb” has been created on the server. The major table that we use for sensor data is called “sensordata”. Its structure is described in the following Table 1.

In the table, “record_id” is an automatic field acting as a primary key. “sensor_type” can be one of the following values: “COFFEEPOT”, “HEADCNT”, and “PROJECTOR”. “sensor_id” identifies the sensor and can be something like “Cam2” and “Cam3”. Whenever an entry is updated, the change time is recorded in the “timestamp” field. Depending on the type of sensor, “value” can be the current coffee pot level, the head

count of a room, or the uptime of a projector. In case the record comes with a file, such as .jpg or .avi for camera data, “has_file” flags this information and the file location can be retrieved by accessing another table.

Field	Type	Null	Key	Default	Extra
record_id	int(10) unsigned		PRI	NULL	auto_increment
sensor_type	varchar(15)	YES		NULL	
sensor_id	varchar(15)	YES		NULL	
timestamp	timestamp	YES		CURRENT_TIMESTAMP	
value	int(11)	YES		NULL	
has_file	tinyint(1)	YES		NULL	

Table 1: The definition of table “sensordata” which stores important sensor data of the SmartCIT system.

The mcamip updates the head count record whenever a person enters or leaves the room. CoffeePotImgProc rewrites the coffee pot entry once the 100% level of the coffee pot is detected. ProjectorImgProc queries the database for head count information periodically. If the head count returns zero, it keeps updating the uptime of the projector. At a typical time, the table can look like:

record_id	sensor_type	sensor_id	timestamp	value	has_file
1	CoffeePot	Cam3	2006-05-18 11:28:07	40	0
2	HeadCnt	Cam2	2006-05-18 11:29:15	0	0
3	Projector	Cam1	2006-05-18 11:29:22	12	0

Table 2: The contect of table “sensordata” at a typical time.

The database makes it possible for disparate system components to collaborate together efficiently.

VI. EMAIL/SMS NOTIFICATION

An email/SMS notification program has been developed to facilitate sending alert messages to pre-registered users when certain events occur. This program maintains a small configuration file (implemented by a plain text file) of contact information. Each database entry is formatted like this:

```
--------
EVENT_NAME
EVENT_ID (to uniquely specify events)
SENSOR_ID (to uniquely specify sensors for the same event)
MESSAGE_FILENAME (located in ./messages/)
DEVICE_TYPE (email/pda)
ADDRESS_OF_RECEIVER
--------
```

The program reads the configuration file when it starts up. When a pre-registered event happens, e.g. the coffee pot is brewing and finally reaches the 100% full status, the daemon system is triggered to call the email notification program to send an alert message to the person who is registered for this event. The notification program then locates the message file stored in ./messages/ and sends the content. In the example of coffee pot being full, a message like "please come and take care of the full coffee pot located in xxx" will be sent to the ADDRESS_OF_RECEIVER specified in the configuration settings.

The C program sends email/SMS by calling the system provided /usr/sbin/sendmail. SMS latency varies from carrier to carrier. For Verizon Wireless

which we tested, it should be within several seconds, providing a very satisfactory performance.

VII. WEB INTERFACE

SmartCIT provides a simple web interface for the public user, which functions as a window into the data collection and query processing framework of the project. There are currently two web pages set up; one to view the output of the coffee pot monitoring unit, and another to view the head counts of various rooms. Located in a web server running apache2, both pages are constructed in HTML and PHP languages with database connectivity. Since real time camera streams can be viewed online, it requires browsers that support Java Runtime Environment (JRE).

The homepage of the SmartCIT web services has links leading to pages devoted to room head count and coffee pot status monitoring. We take the head count service as an example to explain how the pages function. The user reaches the head count page and chooses the room that he/she wants to investigate. After the “update” button is hit, a form is submitted t to the server and the php script gets the room location. The script then sends a query to the database regarding the room head count in that room. When the server returns the number, the script structures it in HTML format and returns to the user. Below are two pictures showing the head count page on the browser window before and after the room location is selected. On the left screen, “Room #1” is selected. After querying the database server, the head count of that room is displayed in the right screen as well as a link to view the real-time camera video.

The page for the coffee pot works similarly. These pages are very simple aesthetically, however, they work fine as a proof of concept.

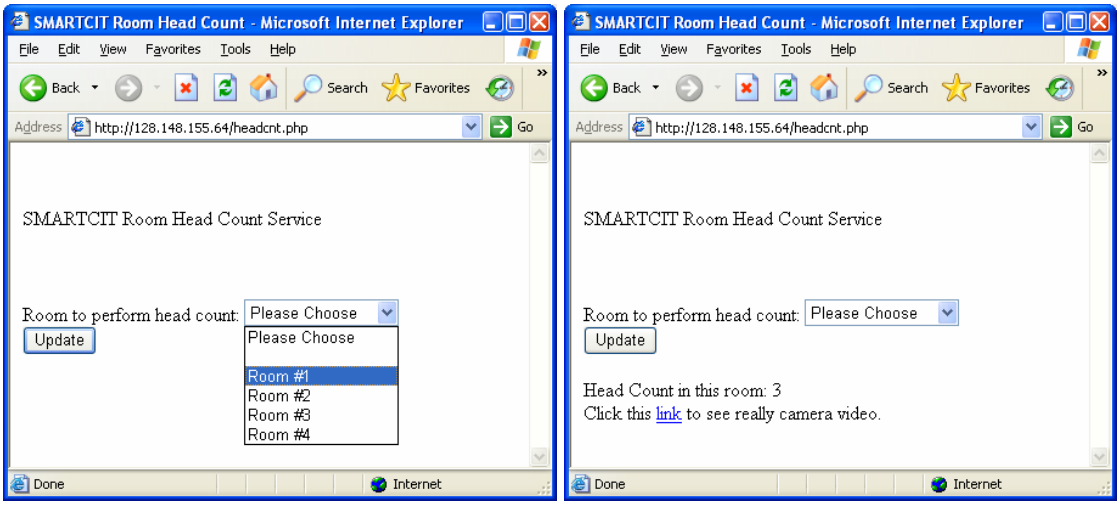


Figure 3: Screen captures that show the head count page before (left) and after (right) the room location is selected.

VIII. CONCLUSION

The SmartCIT program has proven to be a success as a proof of concept, and could be extended to support more diverse data streams and more complex queries easily. The image processing units work well to performance various monitoring and detection tasks and they work with the database and web servers seamlessly. The use of web cameras seems to be a wise decision in that it provides sufficiently good image quality which eventually leads to reliable detection. It would have been more successful if not for the bugs regarding the coordination between MoteTrack software and hardware. The code is straightforward and well-documented enough that a future group could easily pick up and extend this project. As far as this implementation of SmartCIT is concerned, virtually all goals have been met and every milestone reached.

ACKNOWLEDGMENT

The author thanks Prof. Ugur Cetintemel for his advice and help on this project. He also acknowledges Bill Cabral, Kyu-Wook Cho, and Christian Convey for their collaboration work in the early stage of the SmartCIT development.

APPENDIX: USER MANUAL OF THE SMARTCIT SYSTEM

==========

README for the SmartCIT System

Prepared by Vince Liang

vince.liang@gmail.com

Last modified: May 2006

==========

==========

Brief Overview

==========

The SmartCIT project is an attempt to merge information from disparate data streams through a custom query engine to answer interesting and complex queries about the environment and the people in it. The information for these data streams is collected through remote sensors (wired or wireless) strategically dispersed throughout the test space. Currently tested data streams includes personal location information, provided by a modified implementation of Harvard's MoteTrack system, and image processing modules, which analyze data received from web cams. Major functionality of the system includes monitoring coffee pot level, protecting projectors idling overtime, and head counting in a room. The current deployment of this system stands as an interesting proof of concept, and it could be extended in any number of ways.

==========

Hardware Setup

==========

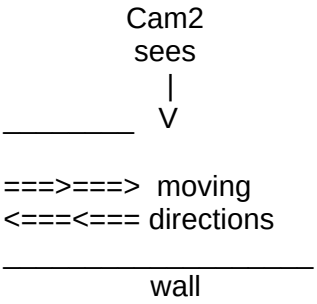
SmartCIT currently uses 3 D-link DCS-900W webcams. We call them Cam1, Cam2, and Cam3. The webcams are all wired to a router which also connects the host computer. All of them need to open port 81 for network communication, which can be set in the configuration page.

Cam1: 192.168.0.11

monitors the projector and should be mounted somewhere near the projector screen.

Cam2: 192.168.0.12

monitors a door of a room and should be mounted so that it sees the incoming and outgoing people from the side.



Cam3: 192.168.0.13

monitors a coffee pot and should be mounted somewhere close to the pot so that it captures the whole pot.

All of the cameras are not password protected.

==========

Software File List

==========

SmartCIT> ll -R

.:

total 18

drwxr-xr-x 6 vincent users 1024 2006-05-18 12:12 cam

drwxr-xr-x 2 vincent users 1024 2006-05-18 13:45 coffeepotimgproc

drwxr-xr-x 4 vincent users 1024 2006-05-18 13:46 event-notify

drwxr-xr-x 2 vincent users 1024 2006-05-18 13:46 projectorimgproc

-rw-r--r-- 1 vincent users 12589 2006-05-18 13:44 README

./cam:

total 56

-rwxr-xr-x 1 vincent users   62 2006-05-17 07:20 cam-frame-md-coffeepot.sh

-rwxr-xr-x 1 vincent users   66 2006-05-17 15:11 cam-frame-md-projector.sh

drwxr-xr-x 3 vincent users 2048 2006-05-17 07:25 camimages

-rwxr-xr-x 1 vincent users   71 2006-05-16 22:08 cam-mplayer.sh

-rw-r--r-- 1 vincent users 1748 2006-05-16 23:07 cam_view.htm

-rwxr-xr-x 1 vincent users   80 2006-05-16 19:56 cam-x.sh

-rwxr-xr-x 1 vincent users 45948 2005-12-20 14:31 mcamip

drwxr-xr-x 3 vincent users 1024 2006-05-18 13:46 mcamip-0.7.3

drwxr-xr-x 2 vincent users 1024 2006-05-17 15:12 projectorimgs

./cam/camimages:

total 8

-rw-r--r-- 1 vincent users   3 2006-05-17 13:13 jpeg_sequence_number

-rw-r--r-- 1 vincent users 7126 2006-05-17 13:13 smartcitcam01.jpg

./cam/mcamip-0.7.3:
total 232
-rw-r--r-- 1 vincent users 7433 2006-05-16 20:57 alpha_num.c
-rw-r--r-- 1 vincent users 8016 2006-05-18 12:05 alpha_num.o
-rw-r--r-- 1 vincent users 2660 2005-12-20 14:31 CHANGES
-rw-r--r-- 1 vincent users 17976 2005-12-20 14:31 LICENSE
-rw-r--r-- 1 vincent users 29027 2006-05-17 04:35 Makefile
-rwxr-xr-x 1 vincent users 42260 2006-05-18 12:05 mcamip
-rw-r--r-- 1 vincent users 1100 2005-12-20 14:31 mcamip-0.7.3.lsm
-rw-r--r-- 1 vincent users 23896 2006-05-18 13:43 mcamip.c
-rw-r--r-- 1 vincent users 2574 2006-05-17 04:46 mcamip.h
-rw-r--r-- 1 vincent users   33 2005-12-20 14:31 mcamip.man
lrwxrwxrwx 1 vincent users   10 2006-05-18 12:05 mcamip._man -> mcamip.man
-rw-r--r-- 1 vincent users 19920 2006-05-18 12:05 mcamip.o
-rw-r--r-- 1 vincent users 1865 2005-12-20 14:31 mcamip_proto.h
-rw-r--r-- 1 vincent users 5722 2005-12-20 14:31 README
-rw-r--r-- 1 vincent users 14850 2005-12-20 14:31 txtfont.h
-rw-r--r-- 1 vincent users 21009 2006-05-18 13:44 video.c
-rw-r--r-- 1 vincent users 13572 2006-05-18 12:05 video.o
-rw-r--r-- 1 vincent users 2934 2005-12-20 14:31 x11.c
-rw-r--r-- 1 vincent users   261 2005-12-20 14:31 x11.h
-rw-r--r-- 1 vincent users 3648 2006-05-18 12:05 x11.o

./cam/projectorimgs:
total 10
-rw-r--r-- 1 vincent users   4 2006-05-17 17:21 jpeg_sequence_number
-rw-r--r-- 1 vincent users 8535 2006-05-17 17:21 smartcitcam01.jpg

./coffeepotimgproc:
total 37

-rw-r--r-- 1 vincent users 484 2006-05-18 13:21 CoffeePotImgProc$1.class
-rw-r--r-- 1 vincent users 5994 2006-05-18 13:21 CoffeePotImgProc.class
-rw-r--r-- 1 vincent users 14285 2006-05-18 13:33 CoffeePotImgProc.java
-rw-r--r-- 1 vincent users 4257 2006-05-18 13:21 CoffeePotImgProc$LoadImageThread.class
-rw-r--r-- 1 vincent users 3672 2006-05-18 13:21 CoffeePotImgProc$ML.class
-rw-r--r-- 1 vincent users 2368 2006-05-18 13:21 CoffeePotImgProc$UpdateFileThread.class
-rw-r--r-- 1 vincent users 2225 2006-05-18 13:21 ImgPixelGrabber.class

./event-notify:
total 42
-rw-r--r-- 1 vincent users 565 2006-05-18 13:35 eventDB.db
-rwxr-xr-x 1 vincent users 31512 2006-05-18 13:25 event-notify
-rw-r--r-- 1 vincent users 5232 2006-05-18 13:37 event-notify.C
-rw-r--r-- 1 vincent users 349 2005-11-30 17:46 Makefile
drwxr-xr-x 3 vincent users 1024 2006-05-18 12:08 messages
-rwxr-x--- 1 vincent users 99 2005-11-30 17:51 run-sim

./event-notify/messages:
total 4
-rw-r----- 1 vincent users 91 2006-05-17 10:17 coffeepot1.msg
-rw-r--r-- 1 vincent users 124 2006-05-17 10:19 projector1.msg
-rw-r----- 1 vincent users 26 2005-12-20 14:32 sample
-rw-r----- 1 vincent users 58 2005-12-20 14:32 ta3.msg

./projectorimgproc:
total 39
-rw-r--r-- 1 vincent users 2225 2006-05-17 21:00 ImgPixelGrabber.class
-rw-r--r-- 1 vincent users 484 2006-05-17 21:00 ProjectorImgProc$1.class

-rw-r--r-- 1 vincent users 6454 2006-05-17 21:00 ProjectorImgProc.class
-rw-r--r-- 1 vincent users 15151 2006-05-18 13:33 ProjectorImgProc.java
-rw-r--r-- 1 vincent users 4731 2006-05-17 21:00 ProjectorImgProc$LoadImageThread.class
-rw-r--r-- 1 vincent users 3673 2006-05-17 21:00 ProjectorImgProc$ML.class
-rw-r--r-- 1 vincent users 3058 2006-05-17 21:00 ProjectorImgProc$UpdateFileThread.class

Files stored in the web server root directory:
-rw-r--r-- 1 vincent users 1746 2006-05-17 12:34 cam1_view.htm
-rw-r--r-- 1 vincent users 1746 2006-05-17 06:52 cam2_view.htm
-rw-r--r-- 1 vincent users 1543 2006-05-17 12:43 coffeepot.php
-rw-r--r-- 1 vincent users 1494 2006-05-17 12:41 headcnt.php
-rw-r--r-- 1 vincent users 416 2006-05-18 11:56 index.php

==========
Compile the Source Code
==========

The source code is distributed in various directories. It is eary to compile them since makefiles has been written.

SmartCIT/cam/mcamip-0.7.3> make
SmartCIT/projectorimgproc> javac Projectorimgproc.java
SmartCIT/coffeepotimgproc> javac CoffeePotImgProc.java
SmartCIT/event-notify> make

==========

Database Configuration

==========

SmartCIT uses mysql database (version 4.1.13).

username: smartcit
password: smartcit

After installation and initial account setup, start the server:
shell> bin/mysqld_safe &

If the server starts successfully, the following information is displayed:
shell> mysqladmin version
mysqladmin Ver 8.41 Distrib 4.1.13, for suse-linux on i686
Copyright (C) 2000 MySQL AB & MySQL Finland AB & TCX DataKonsult AB
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL license

Server version	4.1.13
Protocol version	10
Connection	Localhost via UNIX socket
UNIX socket	/var/lib/mysql/mysql.sock
Uptime:	20 sec

Threads: 1 Questions: 2 Slow queries: 0 Opens: 11 Flush tables: 1 Open tables: 0
Queries per second avg: 0.100

To create a database named smartcitdb:
shell> bin/mysql --user=smartcit -p

Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 4.1.13

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE DATABASE smartcitdb;

To make smartcitdb the current database, use this command:
mysql> USE smartcitdb
Database changed

Use a CREATE TABLE statement to specify the layout of the sensordata table:
mysql> CREATE TABLE sensordata (record_id INT UNSIGNED NOT NULL AUTO_INCREMENT, sensor_type VARCHAR(15), sensor_id VARCHAR(15), timestamp TIMESTAMP, value INT, has_file BOOLEAN, PRIMARY KEY (record_id));

mysql> describe sensordata;

Field	Type	Null	Key	Default	Extra
record_id	int(10) unsigned		PRI	NULL	auto_increment
sensor_type	varchar(15)	YES		NULL	
sensor_id	varchar(15)	YES		NULL	
timestamp	timestamp	YES		CURRENT_TIMESTAMP	
value	int(11)	YES		NULL	
has_file	tinyint(1)	YES		NULL	

6 rows in set (0.00 sec)

Use the following statments to insert some initial records:

```
mysql> insert into sensordata value ('', 'CoffeePot', 'Cam3', CURRENT_TIMESTAMP, 0, 0);
mysql> insert into sensordata value ('', 'HeadCnt', 'Cam2', CURRENT_TIMESTAMP, 0, 0);
mysql> insert into sensordata value ('', 'Projector', 'Cam1', CURRENT_TIMESTAMP, 0, 0);

mysql> select * from sensordata;
+------------+----------------+-------------+---------------------------+--------+-------------+
| record_id | sensor_type | sensor_id | timestamp                 | value | has_file  |
+------------+----------------+-------------+---------------------------+--------+-------------+
|        1    | CoffeePot    | Cam3       | 2006-05-18 11:28:07 |    0   |       0     |
|        2    | HeadCnt      | Cam2       | 2006-05-18 11:29:15 |    0   |       0     |
|        3    | Projector    | Cam1       | 2006-05-18 11:29:22 |    0   |       0     |
+------------+----------------+-------------+---------------------------+--------+-------------+
3 rows in set (0.00 sec)
```

==========

Web Server Configuration

==========

Apache2 is used for SmartCIT. PHP modules with mysql connection capability is necessary to support the system provided web services. No additional work is required for the web server.

==========

How to Start the System

==========

Assuming that the cameras are already properly wired and mounted and the database/web servers are already up running, following the described steps will start the whole system. Programs are recommended to run in individual consoles as they might have information output to stdout/stderr.

Assume the current working directory is the SmartCIT root directory.

Cam1:

SmartCIT/cam> ./mcamip -c -e 1000 -t -j -d ./projectorimgs -a 192.168.0.11 -p 81

SmartCIT/coffeepotimgproc> java -classpath .:/usr/share/java/mysql-connector-java-3.1.8.jar CoffeePotImgProc

Cam2:

SmartCIT/cam> ./mcamip-0.7.3/mcamip -c -e 5000 -a 192.168.0.12 -p 81 -y | mplayer -

Cam3:

SmartCIT/cam> ./mcamip -c -e 1000 -t -j -d ./camimages -a 192.168.0.13 -p 81

SmartCIT/projectorimgproc> java -classpath .:/usr/share/java/mysql-connector-java-3.1.8.jar ProjectorImgProc

==========

Event Notification Configuration

==========

The configuration of event notification is stored in file SmartCIT/event-notify/event.db. The file is so structured for easy maintenance.

NAME-OF-EVENT
event id (to uniquely specify events)
sensor id (to uniquely specify sensors for the same event)
message file name (located in /home/vincent/work/SmartCIT/messages/)
type of device (email/pda)
address of receiver

A sample event.db entry looks like:

COFFEEPOT-READY
1
3
coffeepot1.msg
email
vince_liang@brown.edu

Text messages going to cell phones need to be sent carrier-specific addresses. E.g. for verizon wireless number 401-1234-567, the receiving address is 4011234567@vtext.com.