We present a prototype web application for the Scalable Medical Alert and Response Technology (SMART) [1]. The project aims to wirelessly monitor vital signs and locations of otherwise unattended patients in the waiting area of an emergency department. Waiting patients would feel secure that their condition is being monitored even when a caregiver is unavailable. Caregivers will be alerted in real-time to problems occurring in the waiting room, while threshold values for patient alerts and priorities can be dynamically adjusted. We hope to gain some insights from deployment in the emergency room settings, which could be extended to disaster situations where patients drastically outnumber available caregivers. This application is currently in development and will be tested in actual emergency rooms in the near future.

**SMART Central system description**

The system links caregivers, patients, a streaming database that continuously updates patient status, and a processor that analyzes the patient data for alarm conditions. Alarm conditions include high heart rate, low heart rate and low SpO₂ (blood oxygenation level). When an alarm condition is detected, it is dispatched to an available caregiver. Caregivers each have a PDA (HP iPAQ) that allows them to see the roster of patients and to click through to see a patient's vital signs in real-time. Patients wear a similar PDA as an interface for vital signs monitoring (EKG leads and oximetry sensors to measure SpO₂). A cricket location system tracks the patients and caregivers. [2]

The web application facilitates patient monitoring for caregivers by providing a GUI to display relevant information of registered patients and physicians. The main page at [http://nms.lcs.mit.edu/~cnorahs/smart-main.cgi](http://nms.lcs.mit.edu/~cnorahs/smart-main.cgi) is the portal for relevant information on physicians and patients registered with SMART Central, and it:

1. Display a roster of all the patients with their names, ID numbers, age, gender, ESI (Emergency Severity Level), heart rates, and SpO₂.
2. Search the patient database by last name, age range, or ESI.
3. Register new patients and store their information in the PostgreSQL database.
4. De-register patients and remove them from the database.
5. Display all the physicians with their names, ID, and locations
6. ~ 8. Search, register and de-register physicians
9. ~ 10. User surveys for both physicians and patients.

Clicking on the “All Registered Patients” link displays as below:
The text, tables and buttons were coded in HTML. The CGI scripts for sorting, searching, and communicating with the PostgreSQL database were written in Python.

Clicking on the column header buttons sorts the roster by that column. The text to be sorted is passed as a hidden string variable to the reloaded HTML page, where the CGI script would parse the string and display it in a table format. The figure above comes up after clicking on the “Last Name” button.

The columns for heart rate (HR) and SpO2 automatically update at preset intervals (i.e. 30 seconds). The data arrives in continuous streams from the patient’s sensor leads (or the PostgreSQL database files in the simulated case). To enable communication between the data processor and the monitoring device (or database), we import the Python socket module into the CGI script. The CGI for the webpage sends a query to the streaming database or a patient’s iPAQ, which returns data streams packaged and interpreted by the Python struct module. The resulting data is displayed on the patient information webpage. In this example, all the patients are simulated. Patient #2 (John Jimm) has streaming data set up (87 beats/min and 97% SpO2). Software development for the data stream processing was based on source code by Jason Waterman.

The automatic update function is coded by JavaScript embedded in the Python CGI. The JS also codes the time counter for the last update.

Clicking on one of the last names opens a page with more detailed information about the patient’s alarm conditions, allergies, and current medications.

A login script [3] provides restricted access for SMART Central. Only the registered physicians can access the main page and all the links within. The login script is an open-source Python utility written by Michael Foord.

Lastly, there are two online user surveys with free-response and Likert scale questions to gather user feedback from patients and caregivers. The questions were written by the clinical decision group at Brigham’s Women Hospital.

Benefits

This system provides an easy-to-use interface for emergency room caregivers to access patient information from any web terminal. The automatic page refresh ensures that medical data is updated frequently and reliably, essential for an emergency room setting. With the location tracking functionality implemented, physicians could locate patients and other physicians quickly.

Challenges

The first major design challenge encountered was implementing the sorting function. Since HTML is a stateless language, all the information from the previous unsorted page must be passed to the sorted page as a hidden variable. The syntax for hidden variable passing took some exhaustive search and debugging. Next, parsing the hidden variable string for display on the sorted page called for modularized subroutines tested sequentially for robustness.

Secondly, meticulous proofreading was required to combine CGI code in Python with the JavaScript for automatic page refresh and time stamps. This was equivalent to making a Python program generate JavaScript, and involved much quotation matching for displaying strings.

Another challenge was getting the streaming data from the Python socket module to work together with web page refreshes. When the browser sends a query to the monitoring device, the device would return data to be interpreted by the Python struct module and displayed on the browser. This data transfer would result in a few seconds of delay, making the automatic page reload sometimes cumbersome. Since speed and accuracy are crucial for the web application to work in clinical settings, work is underway to speed up communication between the browser and the monitoring devices. Simulations had shown the information to be accurate yet not as efficient as desired.

Alternative Implementation Methods

Instead of Python, Perl, C, or Java could also be used. Python is a powerful language with versatile string manipulation functions, ideal for parsing strings in this web application. Furthermore, its clean syntax was easier to debug and decreased development time. However, Java or C might be more compatible with the embedded JavaScript.

Further Developments

• Incorporate location tracking of caregivers as a search query for patients who need specialized assistance
• Increase performance efficiency of the streaming PostgreSQL database

Acknowledgements

This prototype was developed under the supervision of Dorothy Curtis and Prof. John Guttag.

References