Patient Service System - Project Report

Rohit Saraf	Ashish Mathew	Vivek Madan	Pritish Kamath
08005040	08D10035	08005035	08005004

Group 13

April 13, 2011

under the guidance of
Prof. Kavi Arya

Embedded and Real-Time Systems Laboratory
Department of Computer Science and Engineering
Indian Institute of Technology
Bombay

Contents

Abstract

The project aims at helping the patients and hostipal management. The patients often require 24-hour attendance, which can get very difficult to manage in a large hospital which has a small nursing staff. The project demonstrates usage of *automatic serving bots* using *Firebird V robot* which can answer patients' requests throughout the day. The patients can summon these bots in a similar way as they summon doctors or nurses.

We have demonstrated that patients can use simple, off-the-shelf devices such as TV remote to give requests. The server provides a real-time service with guarantee of no-deadlock/race conditions. The server uses a full fledged database (MySQL) to maintain the queue and manages the bot wisely using a perfectly scalable design. The bots are fair to all the patients. The bot notifies the guards if it is blocked by an SMS.

1 Introduction

Patients in hospitals often require 24-hour attention for their basic needs. However attendants can't be available for patients at all times. Also it may often happen that because of hectic hospital schedules, attendents may forget to give medicines to patients on time. This induces a need for an automatic attendent who could possibly replace human attendants for almost all routine tasks, thus providing more reliability and reducing a tremendous amount of work-load on nursing stations.

The automatic attendent can serve patients' requests such as supplying water or patient-specific medicines and it would be summoned using an easy to use interface by the patients. Possibly the hospital buzzer interface can be integrated with the bot-summoning controller. The attendent service should be real-time. It should serve all patients in a fair manner (i.e. any patient request should have a bounded wait)

2 Hardware Architecture

- Firebird V bots serve as the patient attendants. Each bot has a Zigbee module to communicate to the server. The bot has to deliver resources from the *Store* to the correct patient. The server completely takes control of each bot and gives it instructions to move in the hospital.
- The hospital arena has been simulated with a white line based arena **??**. The patient bot follows the white lines according to the instructions given by the server. The bot uses 3 white line sensors and uses a 7-fold line following algorithm to ensure robust line following (with error correction and error recovery modes). The bot also uses it's front IR sensors to detect obstacles blocking it's path, and informs the server about the block.
- The patient request sensor is a TSOP sensor on a Firebird V bot. It uses a *protocol independent learning based algorithm* to make the TV remote button disambiguation robust.

3 Software Architecture

3.1 Server Side Architecture

- **pss.configuration.Configure**
  This manages the configuration settings for the project. The file config/config.properties is read(which contains all the settings necessary for the project to run successfully.
  This file ensures that changes in database/Zigbee port do not require recompilation/source code edit

- **pss.serialcomm.CommunicationAPI**
  This makes an interface using the Serial communication library called Zigbee, which uses a shared library using the Java JNI architecture. This provides access to the Zigbee just like any other file. This requires many system settings to be done which is explained below.

- **pss.server.Bot**
  Collection of all the information that needs to be accessed for a given bot.

- **pss.server.Graph**
  This contains the graph structure and the search algorithm. The graph can be changed if the arena needs to be changed. It is the first place one might want to change!

- **pss.server.Position**
  This is a class which stores the coordinates and orientation of every Bot. It should be noted that both coordinate and orientation are needed to control the bot.

- **pss.server.RequestHandler**
  This is the main class that handles the interrupts due to receipt of new messages and also invokes the PollingThread to poll when the appropriate time comes.

- **pss.server.Utils**
  This contains some low level bit related calculations to decide the message that has to be sent to the bot.

- **pss.server.database.DBHandler**
  Exposes the Database API's to save patient information and the requests of the patients along with necessary details.

- **pss.server.scheduling.PollingThread**
  When asked, it polls the bots and gets their responses. If required it forwards the responses to the RequestHandler

- **pss.server.test.***
  Contains the testing code that was used to verify that our model/server code is correct. This can be further used for checking extra utils that are added to the project.

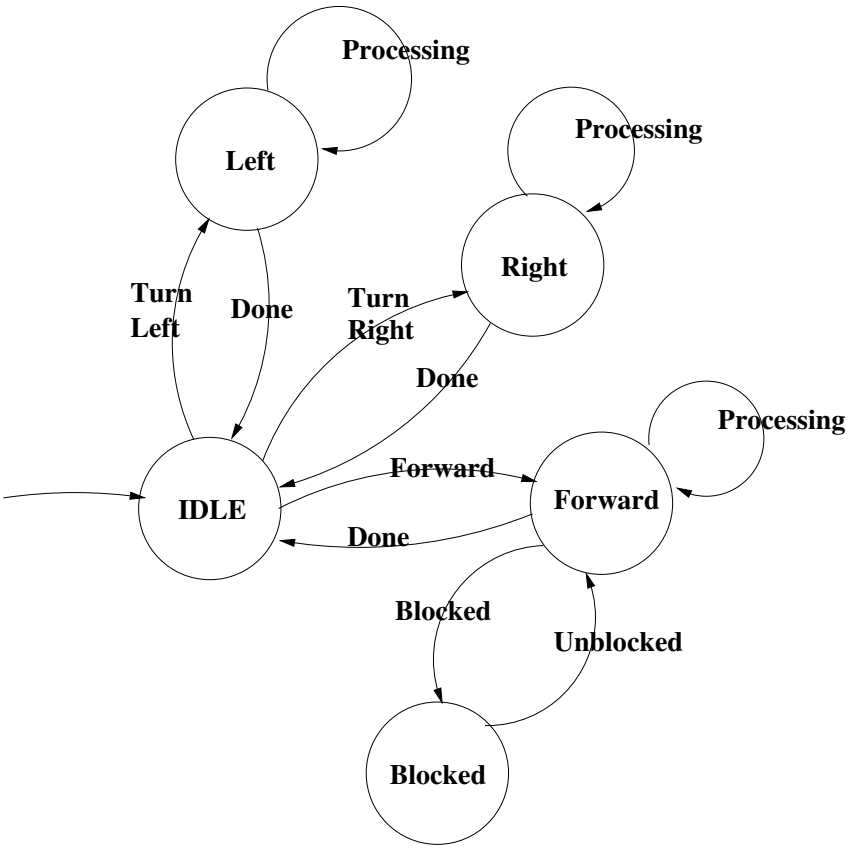

Figure 1: Service Bot FSM

3.2 Finite State Machine

3.3 Bot Side Architecture

- **Bot.c**
  The Bot(s) controller code. This code communicates with the server to make patient service possible.
- **IR.c**
  Remote controller interface which tunes as per the requirements of the remote protocol.
- **Bot_motion.h**
  Provides an API for handling bot locomotion in a safe and correct manner following the white line and avoiding collisions.

4 Work Division

Work/Critical Tasks	**People responsible**
Zigbee communication with Java	Vivek Madan, Rohit Saraf, Ashish Mathew
Making Arena Perfecting Line Follower	Pritish Kamath, Rohit Saraf
Implementing Server Side Polling based Communication Protocol	Vivek Madan, Ashish Mathew
Implementing Bot Side Polling based Communication Protocol	Pritish Kamath, Rohit Saraf
MySQL Database for patients Bot Locomotion Algorithm (server) Simulation	Vivek Madan, Ashish Mathew
Taking input from TV remote Sending SMS via Python Script (internet)	Pritish Kamath, Rohit Saraf,
Configure Scripts and Makefile	Rohit Saraf
Documentation	All

5 Innovations, Challenges and Solutions

5.1 Innovations and Challenges

- Server runs in Ubuntu. Bot AVR-programming done in Ubuntu.
- Zigbee communication using *Java*.
- MySQL database. Makes the project scalable for large number of patients.
- Correction and Recovery while following the white line. Smooth running of the bot, (without any extra squares added at intersections).
- Polling based communication implemented to avoid interference between serving bot and patient bot.
- Maximum parallelism and concurrency to avoid time/resource wastage. (Multithreading which is not concurrent like pthread, but actually parallel).
- Testing everything and making the design modular so that the different parts have least to do with one another. Done using virtual simulation of bots.
- Ensuring no deadlock between multiple bots.
- Getting TV remote to work using a Protocol independent learning based algorithm.
- Sending SMS to phone.
- Efficient Scheduling Algorithm.

6 Testing Strategy

6.1 Arena

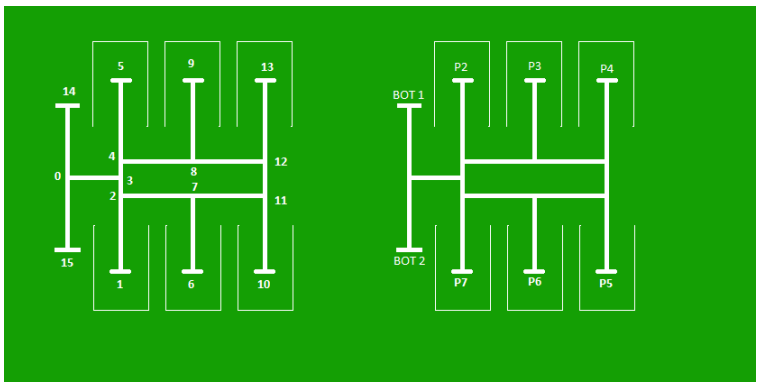

Figure 2: Arena

6.2 Modular Testing

- Testing multiple Zigbee client - single Zigbee server protocol.
- Virtual bot simulation and testing of server.
- Testing for deadlocks and race conditions.
- Server independent testing of serving bots.
- Independent TV remote testing.

6.3 Combined Testing

- Correct patient-request is served by the attendant.
- Attendant stops in presence of obstruction and beeps in case of prolonged obstruction.
- 6 patient controllers using TV remote.
- Simultaneous requests allowed (served in fair-share FIFO order).

7 Code Reusability

1. Server code : Completely object oriented (Java).
2. Modular Design : Modules independent of each other, hence can be changed without changing others.

- A different arena can be stored by changing *Graph.java*.
- Polling protocol can be changed by changing *PollingThread.java*.
- *Properties* file stores all user dependent constants that can be changed without going through/compiling the code.

3. Server is bot-hardware independent. Relies on only a serial communication protocol (not necessarily Zigbee) with the bot.

8 Requirements

8.1 Hardware

- 2 x Firebird V
- 3 x Zigbee modules
- 1 x TSOP Sensor
- 1 x TV Remote

8.2 Software

- avrdude
- avr-libc
- gcc-avr
- JDK
- MySQL
- Python

9 Getting started the code

(Valid for linux systems only)
To execute the code for the first time, run the install script as

sudo ./install
(requires proxy settings)

This will install all the dependencies and drivers required for the project and generate a Makefile.

9.1 How does the install script work

The install script basically uses opensource tools such as automake and autoconf to check for project dependencies. Look at references/man pages for more details. Apart from this it also installs a library for use by the Java JNI architecture. Those scripts are pretty simple and are present in the INSTALL folder.

9.2 Post-Installation Usage

Note :
1.The properties file config/config.properties might require a change. (e.g. Zigbee COM port, Database settings)
Then to

Required Action	Command
compile the server code	make server
To run the server code	make run
To compile and program the serving bot no 1.	make bot
To compile and program the serving bot no 1.	make bot1
To compile and program the patient_IR handler bot	make patient
To clean	make clean
To open the documentation	make doc

10 References

- NeX robotics
- AVR on Ubuntu
- ATmega2560 datasheet
- FireBird V Software Manual
- FireBird V Hardware Manual
- Automake, Autoconf Manuals
- Java JNI
- Zigbee Manual