Requirement Analysis

Design and implement an object oriented application architecture for the simulated procedure of an elevator or a group of several elevators.

First matter is the UI. For such an application, user interface is all it about. Such an application should be developed as a game in my opinion. Thus, we choose Qt 4 for our development which can provide much more convenience in developing an animation-oriented UI application.

To simulate the occasion of heavy elevator traffic is obviously a difficult requirement, because if we use the method of payload testing to make it, the program will no longer be interesting. In order to implement the project with more interactivity. We capture the request of user by a simple HTTP server modeled class. Thus, all the devices including laptop and smart-phones in the same network can make easy control of the request. So will get more entertainment from the application.

As for the performance of such a elevator system, we set a goal that the system will carry the passengers to the destination in the fastest way. That means not we will make the time each passenger waiting for will be the shortest but to shorten the distance the elevators covered. Assuming that the speed of an elevator will not change when it is working, the fastest way to all the passengers must be the shortest way the elevator is running.

Architecture Design of Simulated Elevator

BUPT SSE 2011

Requirement Analysis

Design and implement an object oriented application architecture for the simulated procedure of an elevator or a group of several elevators.

Development Tools

Qt 4 SDK

Qt has much more convenience in UI development especially about animation (so does WPF)

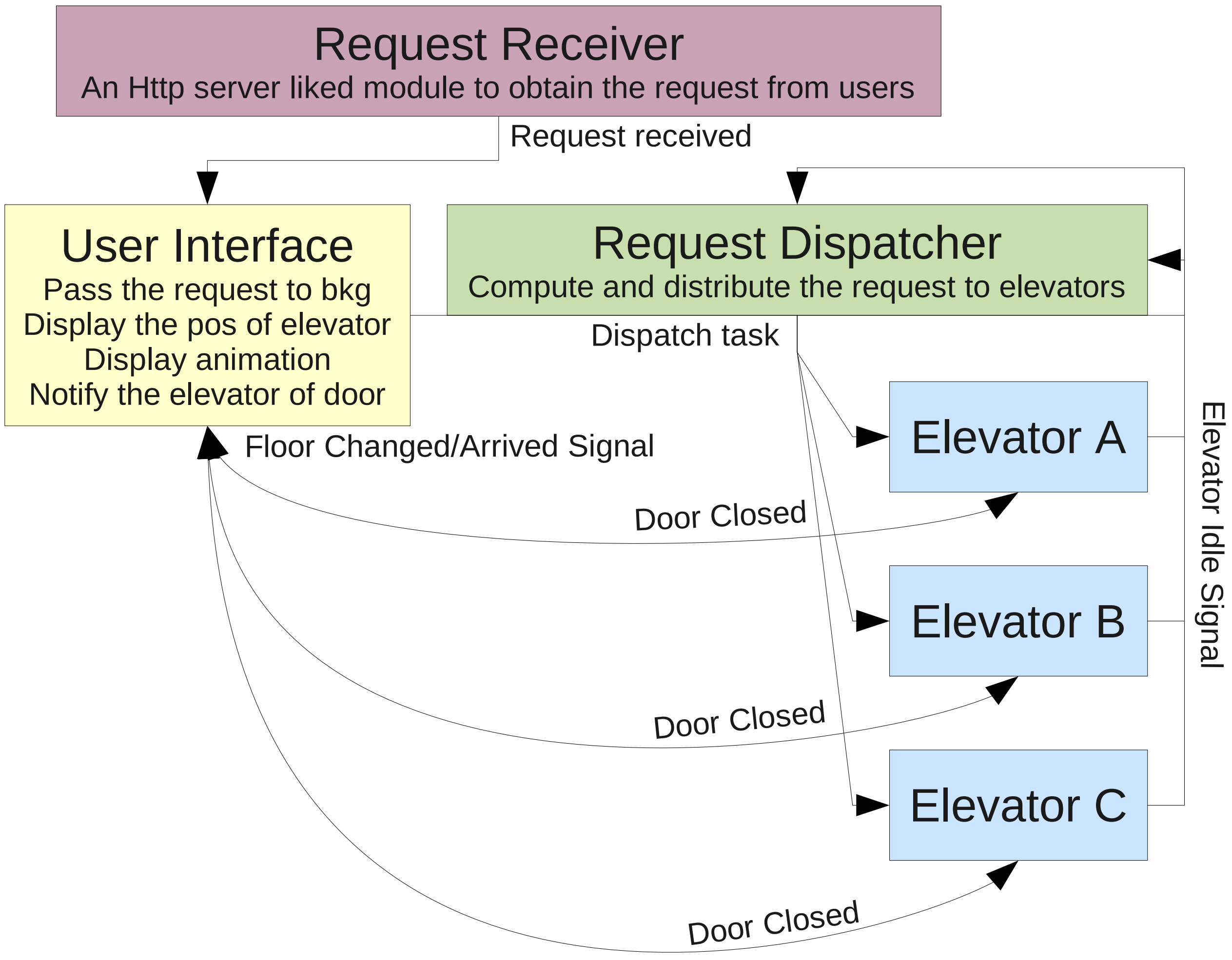
Subversion for VCS

Program Architecture Design

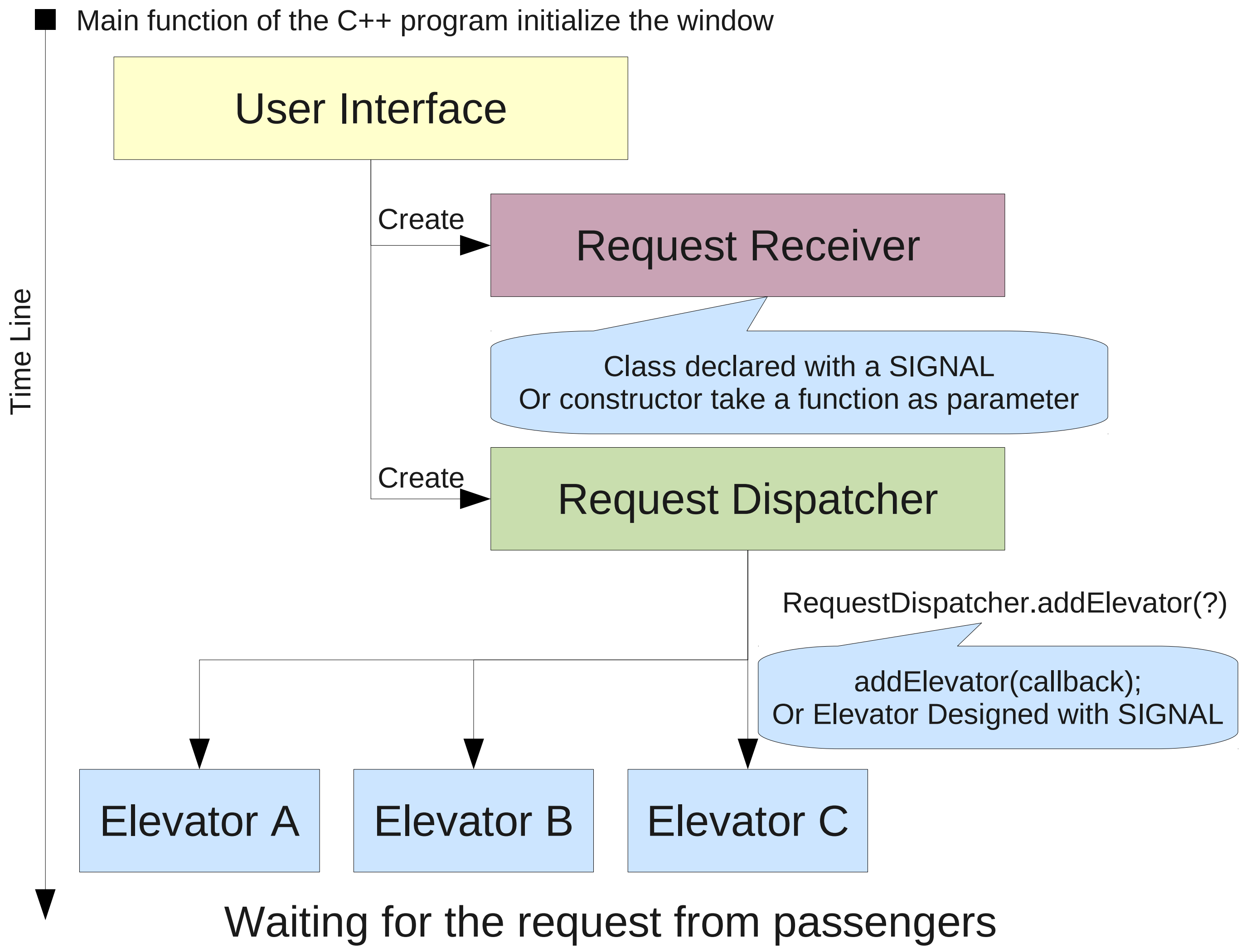
Assuming the occasion of taking an elevator, we found it efficient to use a key-value pair to representing a passenger's departure floor and destination floor. That is, assuming a passenger depart from floor 2 to floor 8,we can represent it in the code by

(2 , 8)

Then, we have the following Classes to finish the simulation of the elevator and elevator group:



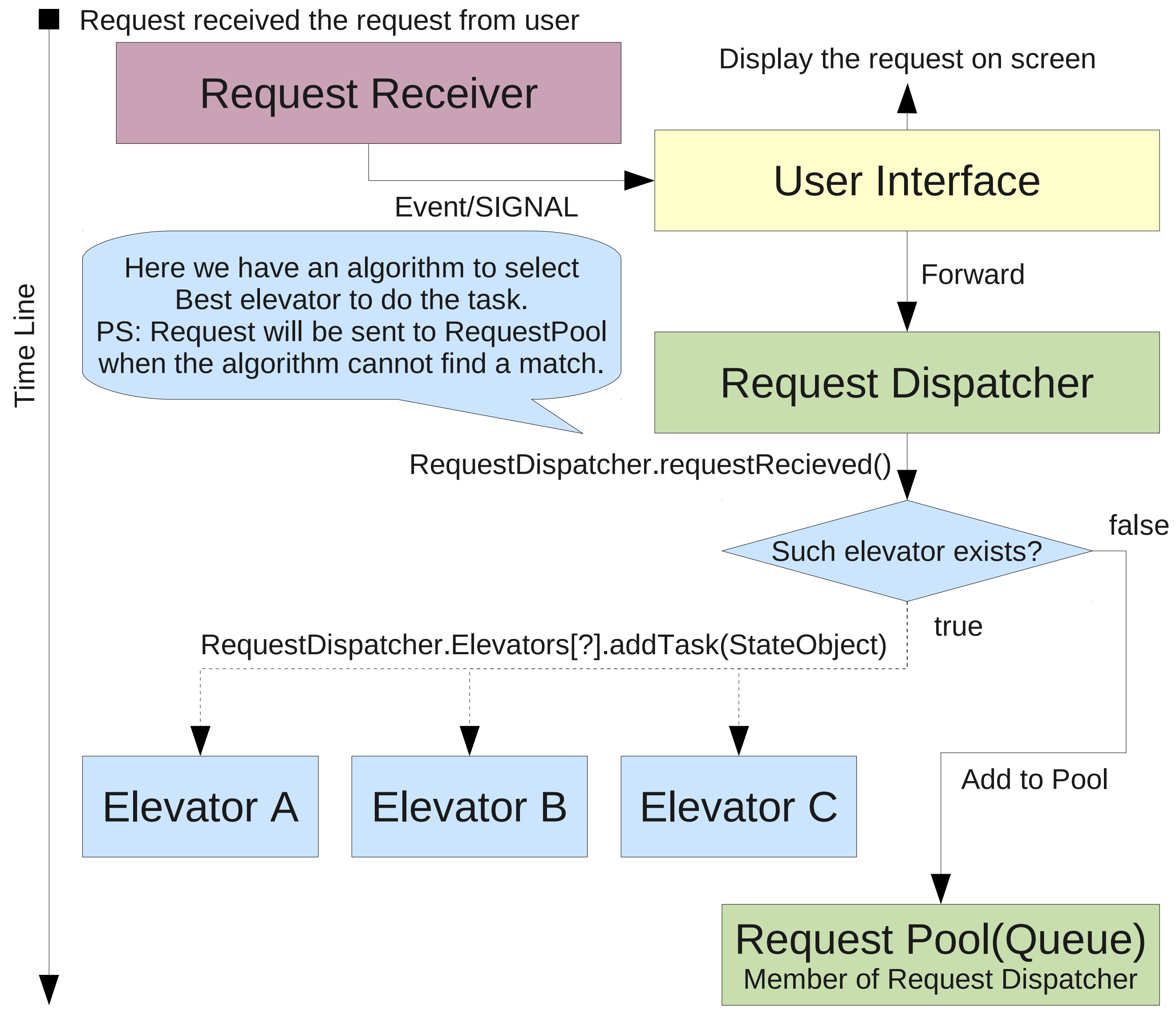
Working Flow and Interfaces


Main function of the C++ program initialize the window
Time Line
User Interface
Create
Request Receiver
Class declared with a SIGNAL
Or constructor take a function as parameter
Create
Request Dispatcher
RequestDispatcher.addElevator(?)
addElevator(callback);
Or Elevator Designed with SIGNAL
Elevator A
Elevator B
Elevator C

Waiting for the request from passengers

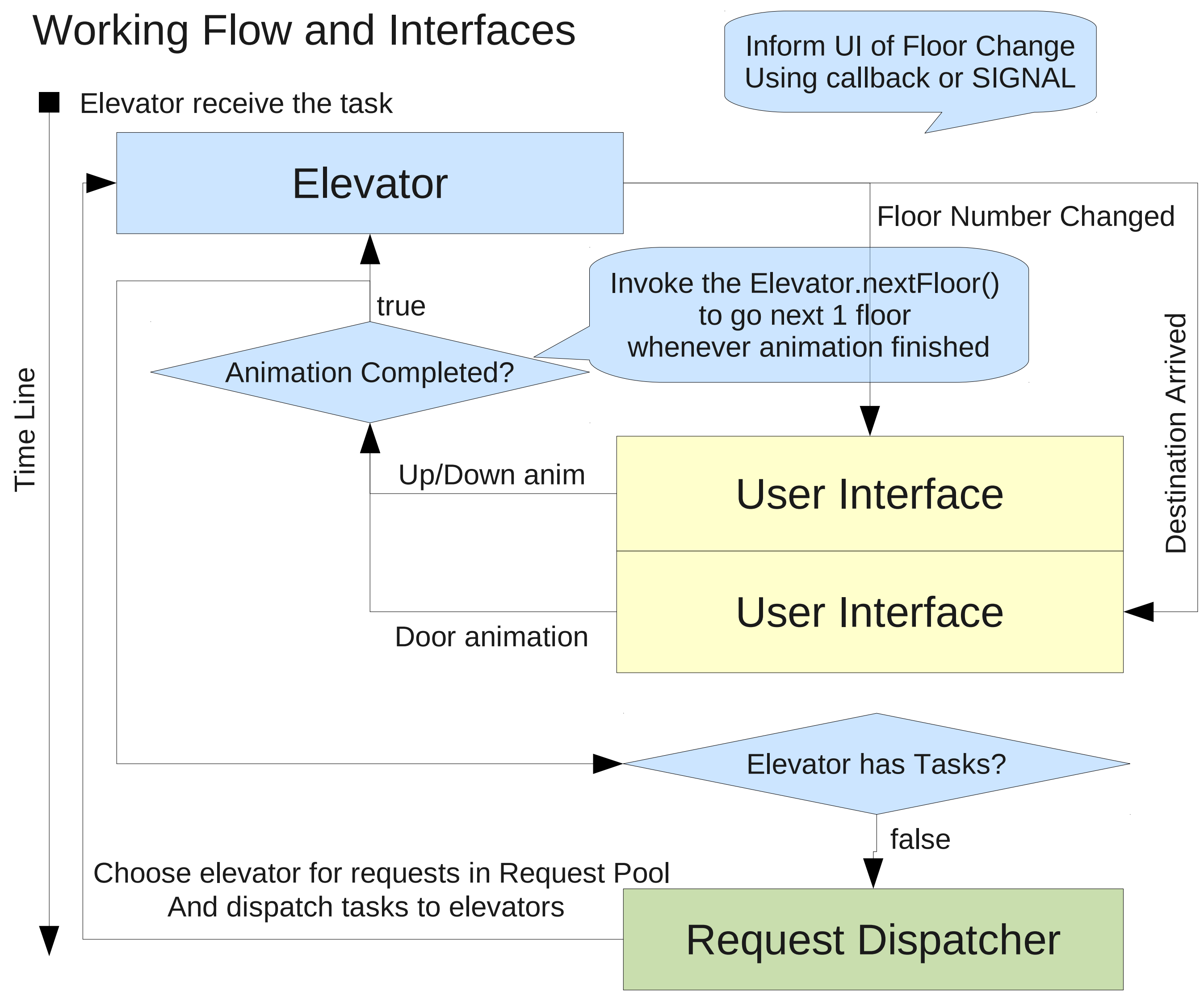
Please turn to next page...

Working Flow and Interfaces


Request received the request from user
Request Receiver
Display the request on screen
User Interface
Event/SIGNAL
Here we have an algorithm to select Best elevator to do the task. PS: Request will be sent to RequestPool when the algorithm cannot find a match.
Forward
Request Dispatcher
Time Line
RequestDispatcher.requestRecieved()
Such elevator exists?
false
true
RequestDispatcher.Elevators[?].addTask(StateObject)
Elevator A
Elevator B
Elevator C
Add to Pool
Request Pool(Queue)
Member of Request Dispatcher

Please turn to next page...

Working Flow and Interfaces


Inform UI of Floor Change Using callback or SIGNAL
Elevator receive the task
Elevator
Floor Number Changed
true
Invoke the Elevator.nextFloor() to go next 1 floor whenever animation finished
Animation Completed?
Time Line
Destination Arrived
Up/Down anim
User Interface
User Interface
Door animation
Elevator has Tasks?
false
Choose elevator for requests in Request Pool And dispatch tasks to elevators
Request Dispatcher

Please turn to next page...

Working Flow and Interfaces

Interface of Main UI Class

Main UI Class must implement following interface:

```
/**
 * Slot to Receive request from RequestReceiver
 * make changed in UI and forward it to RequestDispatcher
 * PS: use this slot In order to avoid cross-thread manipulation
 * @param request
 * request from-to value
 */
void slotRequestReceived(StateObject request);

/**
 * trigger when floor of the elevator changed
 * @param elevatorIndex
 * the index of the elevator
 * @param floorNumber
 * the floor number now
 */
void slotFloorChanged(int elevatorIndex, int floorNumber);

/**
 * Called when floor arrived FloorChanged will not be triggered when arrived
 * @param elevatorIndex
 * the index of the elevator
 * @param floorNumber
 * the floor number now
 * @param requests
 * array of request for current floor(including from & to the floor)
 */
void slotArrived(int elevatorIndex, int floorNumber, StateObject[] requests);
```

Interface of Request Receiver Class

RequestReceiver must implement following interface:

```
/**
 * signal of receiving a request from passenger
 * @param request
 * pointer to the function Receive callback for handler of request
 */
void signalRequestReceived(StateObject request);
```

Working Flow and Interfaces

Interface of Request Dispatcher

RequestDispatcher must implement following interface:

```
/**
 * Create a new elevator
 * @param elevatorIndex
 * index of the elevator intend to create. If the index has exists, the function return false
 * @return
 * Pointer to the elevator created, null if failed
 */
Elevator* addElevator(int elevatorIndex);

/**
 * Get the elevator with the given index
 * @param index
 * the index of the elevator
 * @return
 * pointer to the requested Elevator
 */
Elevator* getElevator(int index);

/**
 * Triggered after one elevator has finished all its tasks
 * scan requestPool here if there is requests in the pool dispatch them!
 * @param elevatorIndex
 * the index of the elevator
 */
void slotElevatorIdle(int elevatorIndex);

/**
 * Add and dispatch a task to the most appropriate Elevator
 * if no appropriate Elevator can be found, just put the request into requestPool!
 * @param request
 * request from-to value
 */
void requestReceived(StateObject request);

/**
 * Queue for unfortunate requests....
 */
Vector<StateObject> requestPool;
```

Working Flow and Interfaces

Interface of Elevator

Elevator must implement following interface:

```
/**
 * Constructor
 * @param elevatorIndex
 * index assigned and stored for raise signals
 */
Elevator(int elevatorIndex);

/**
 * Add a task to the Elevator
 * @param request
 * request from-to value
 */
void requestReceived(StateObject request);

/**
 * Goto next floor in the list
 */
void nextFloor();

/**
 * returns the floor this elevator stops
 * @return
 * current floor number
 */
int getFloor();

/**
 * get whether the elevator is idle
 * @return
 * if this elevator is idle
 */
boolean isIdle();

/**
 * the direction the elevator current is running
 * @return
 * the direction of the elevator
 */
ElevatorDirection getDirection();

/**
 * the direction enumberation
 */
enum ElevatorDirection {NONE,UP,DOWN};
```

Working Flow and Interfaces

Interface of Elevator

Elevator must implement following interface:

```
/**
 * Triggered after this elevator has finished all its tasks
 * @param elevatorIndex
 * the index of this elevator
 */
void signalElevatorIdle(int elevatorIndex);

/**
 * Trigger when floor of the elevator changed
 * @param elevatorIndex
 * the index of this elevator
 * @param floorNumber
 * the floor number now
 */
void signalFloorChanged(int elevatorIndex, int floorNumber);

/**
 * Trigger when floor arrived FloorChanged will not be triggered when arrived
 * @param elevatorIndex
 * the index of the elevator
 * @param floorNumber
 * the floor number now
 * @param requests
 * array of request for current floor(including from & to the floor)
 */
void signalArrived(int elevatorIndex, int floorNumber, StateObject[] requests);
```

Implementation of the Optimized Dispatch Strategy

In the current architecture, we have the class called RequestDispatcher which is to dispatch the request from passengers to the suitable elevator in the elevator group. To have the elevators work efficient, the key is to have an efficient dispatch map. Next page we will introduce how the elevator run with the tasks dispatched by RequestDispatcher.

Steps for choose elevator

1) Capture a request (f,t).
2) Iterate the list (list item i):
    a) whether i's direction matches request's direction if false, next i
    b) whether i's from-floor is smaller/larger(vary on direction) than i's current floor if true diapatch it and end the iteration, else, next i.
3) if the request have been dispatched end the procedure, else, find all idle Elevators choose one nearest to the requested from-floor and dispatch the request to the Elevator.
4) if the request have been dispatched end the procedure, else, add the request to the end of requestPool.

About using of slotElevatorIdle

When slotElevatorIdle triggered by the Elevator object. Iterate the requestPool sequentally to choose the requests for the elevator.

Implementation of the Elevator Working Flow

After revived the task dispatched by RequestDispatcher, the Elevator will start to work.

The Driver Machination of Elevator

The work of the Elevator is controlled by the animation played in UI not the timer or thread defined in the architecture. After the UI finished the move animation, UI is expected to invoke the nextFloor() method of the Elevator, and the Elevator may continue to run. That is, the whole working is driven by the Qt's Animation Framework. In this way, we can reduce the codes on multi-thread programming and make the code more reliable.

The Usage of Signals

Each Elevator object itself should hold a list of tasks recieved, and remove the task from the task list after the to-floor of the tasks matches the current floor. The signalFloorChanged should be raise when floor number changed but no request arrived. And signalArrived should be raised with a list of the arrived requests(from-floor or to-floor). Once the elevator's task list has been empty, signalElevatorIdle should be raised for recheck remaining tasks.

Remarks

Common State Object Implementation

In order to pass the input data a class for data is needed as common state object. It must be very simple and easy to access. The following one:

```
class StateObject{
        public:
                StateObject(int f, int t):from(f),to(t){}
                int getFrom(){return from;}
                int getTo(){return to;}
        private:
                int from = -1;
                int to = -1;
};
```

Obtain Request From User

To make the project more active, the input from user will be obtained through a socket that handle the HTTP GET with a form of Http://127.0.0.1:8369/2-8.

While initializing the UI, Start the RequestReceiver Class for listening request, and the request:GET f-t HTTP 1.x will be parsed into StateObject(f,t) and passed to UI through the callback function passed to construct RequestReceiver. And UI will display the request received then forward the request to RequestDispatcher.

Use of Qt Framework

Qt Application Framework has extended the direct-callback(callback by passing pointer to a function) to a SIGNAL-SLOT. Functions declared as SIGNAL can be bind to SLOT of QObject by the connect function, using SLOT as callback may have help in the multi-threaded application(do not have to deal with the cross-thread manipulation) . All the direct-callback in this document should be written in SIGNAL-SLOT invoke.

About HTTP on Qt

I'm also not very clear of this field. So here is a simple example we can use:
Http://doc.trolltech.com/solutions/3/qtservice/qtservice-example-server.html

Some Spelling Error

For some historical reason, I often spell receive into recieve so be careful about my code...(Hanenoshino)

Detail about User Interface Design

Simulated Elevator - Credits

BUPT SSE 2011

Group Members(Counter-Alphabetical Seq.)

Zeng Ping 曾　平

Wang JiaHui 王佳慧

Li HongWen 李宏文

Guo XiaoHui 郭晓晖

Chen Jie 陈　杰

Cao YangYu 曹洋毓

Document & Architecture

Chen Jie 陈　杰

User Interface & UX Design

Wang JiaHui 王佳慧

Algorithm Design

Zeng Ping 曾　平

Li HongWen 李宏文

Guo XiaoHui 郭晓晖

Cao YangYu 曹洋毓

TODO = =|||