Universidade do Porto
FEUP Faculdade de Engenharia

# Lift Management System

Agents and Distributed Artificial Intelligence

4th year 1st semester
2020/2021

# Problem Description

Managing the **Lift System** in a big skyscraper is a fairly complex task.

With many requests being made in a relatively short amount of time, optimizing elevator allocation for each call is essential to ensure everyone gets to their destination in a reasonable amount of time.

The objective of this project is to develop a **Multi-Agent** System for the management of a set of elevators in a building.

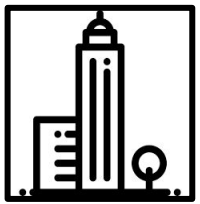
1. Lift icon

2. Building icon

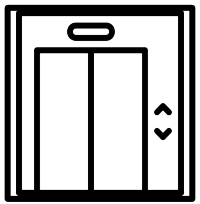
# Agent Schema

Modelling this system as a Multi-Agent System allows us to define 4 distinct agents:

**Building Agent** - Skyscraper. Used to initiate environment.
**Lift Agent** - Building's lifts.
**Floor Panel Agent** - Floor's lift request button.
**Request Agent** - Simulates people in the building.

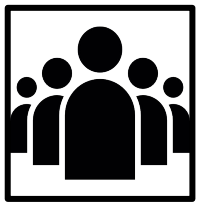


3. Building Agent



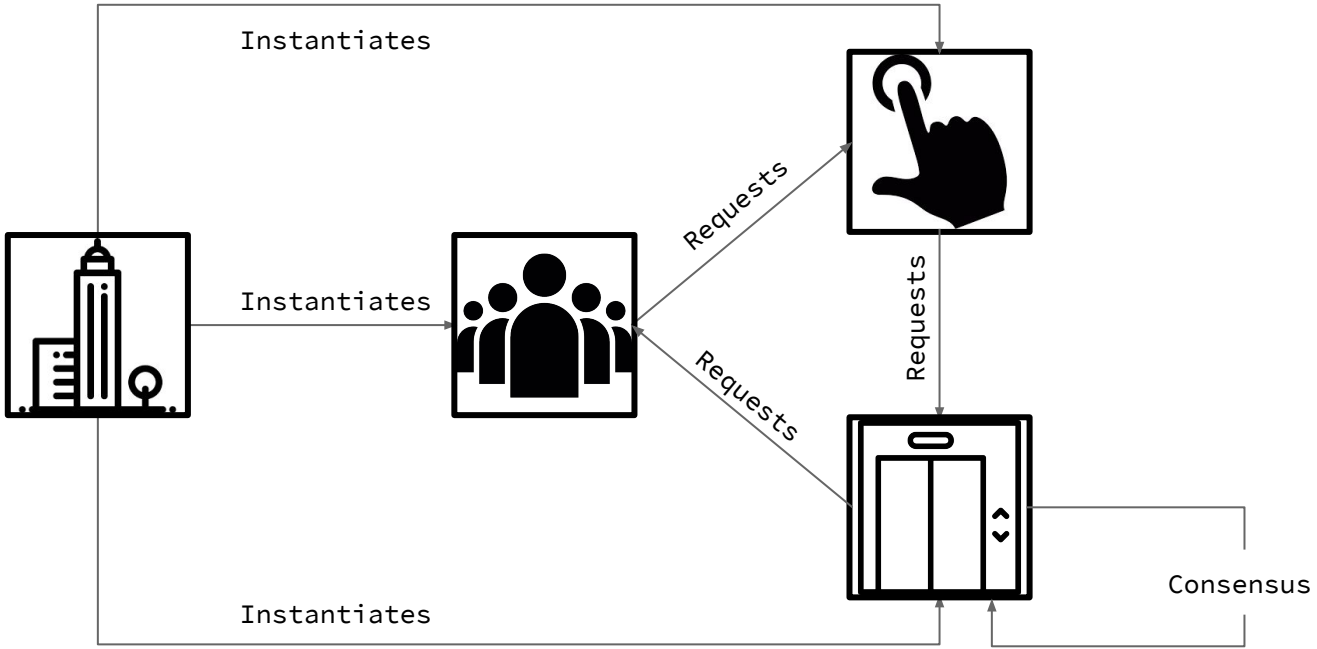
4. Lift Agent

5. Floor Panel Agent



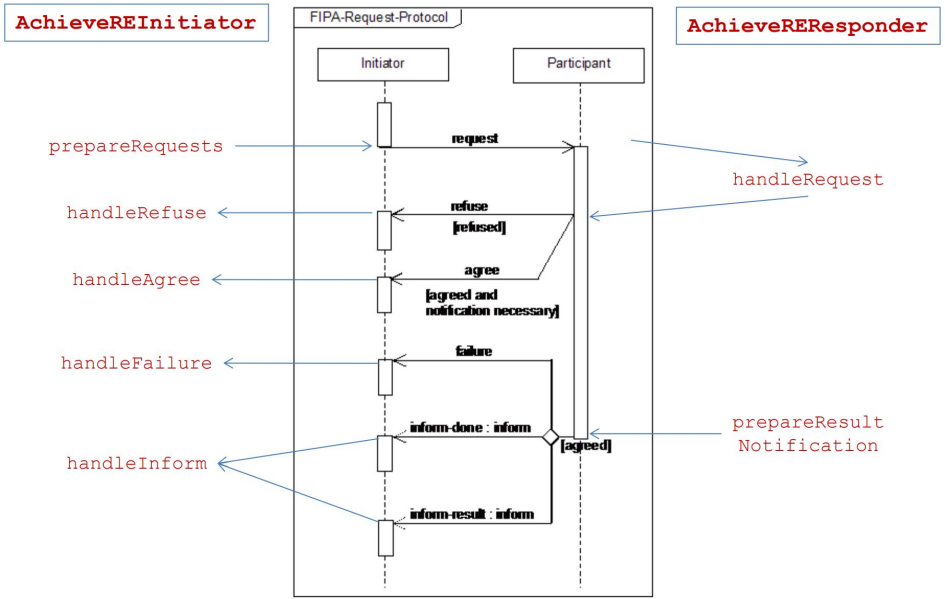
6. Request Agent

# Interactions and Protocols



# Interactions and Protocols

The interaction protocol that is used during communication between Lifts, Floor Panels and Request Agents is the **FIPA Request Protocol.**




7. FIPA Request Protocol's Layout

# Architectures and Strategies

To reach a consensus on which Lift will accept the Floor Panel's request we used a version of the **Bully Election Algorithm**.

To update the Lift's position we used a **Ticker Behavior**.

```
private void processProposalMessage() {
    //Receiving Proposal Messages
    ACLMessage msg = myAgent.receive(templatePropose);
    int liftId = 0;
    float proposedTime = 0;
    if(msg != null) {

        String[] content = msg.getContent().split(":", 2);

        proposedTime = Float.parseFloat(content[0]);
        liftId = Integer.parseInt(content[1]);

        proposalsList.put(liftId, proposedTime);
    }
    //Processing Proposal Messages

    var myProposal = lift.getCurrentLiftProposal();
    //Only if I have proposal and received all other proposals
    if(proposalsList.size() >= lift.getContacts().size() && myProposal != null) {
        boolean betterProposal = true;

        Iterator it = proposalsList.entrySet().iterator();
        while (it.hasNext()) {
            HashMap.Entry pair = (HashMap.Entry)it.next();
            float time = (float)pair.getValue();
            int id = (int)pair.getKey();
            if(myProposal.getTime() > time) betterProposal = false;
            else if(myProposal.getTime() == time && id < lift.getId()) betterProposal = false;
            it.remove();
        }
        if(betterProposal) acceptProposal();
    }
}
```

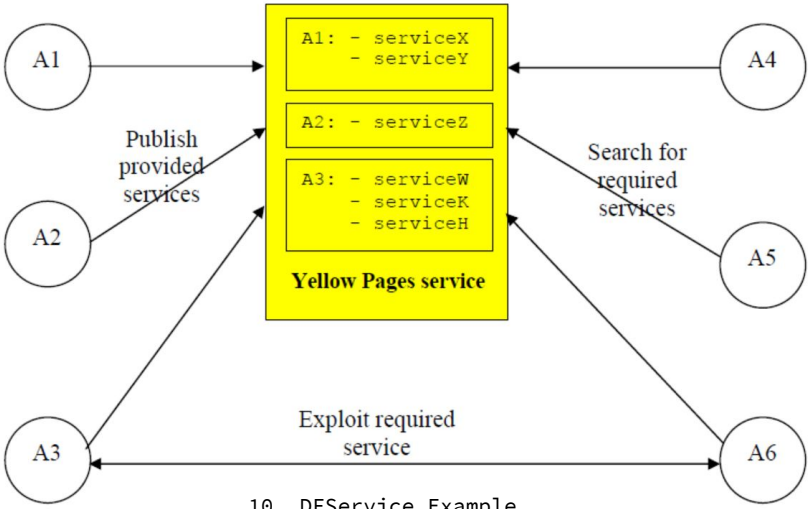
8. Fragment of Bully Election Algorithm

```
public LiftTickerBehaviour(Agent agent, long period) {
    super(agent, period);
    this.myAgent = (LiftAgent) agent;
    this.realAgentPosition = (float) this.myAgent.getFloor();
    this.tick = 0;
}

@Override
protected void onTick() {

    if(!this.myAgent.getTaskList().isEmpty()) {
        if(this.myAgent.getTaskList().get(0).getFloor() == this.myAgent.getFloor()) {
            processRequest();
        }
        else if(this.myAgent.getTaskList().get(0).getFloor() > this.myAgent.getFloor()) {
            this.realAgentPosition += (float) (this.getPeriod() / 1000) / this.myAgent.getSpeed();
        }
        else {
            this.realAgentPosition -= (float) (this.getPeriod() / 1000) / this.myAgent.getSpeed();
        }

        updatePosition();
    }

    this.tick = this.tick + 1;
    if( this.tick % 2 == 0) {
        this.myAgent.writeToFile(tick, this.myAgent.getFloor());
    }
}
```

9. Lift Ticker Behavior

# Other Mechanisms

To know how many Lifts are present in the system, upon their creation, they all subscribe to the lift-service using the **Directory Facilitator Service**. Floor Panel have access to this information.



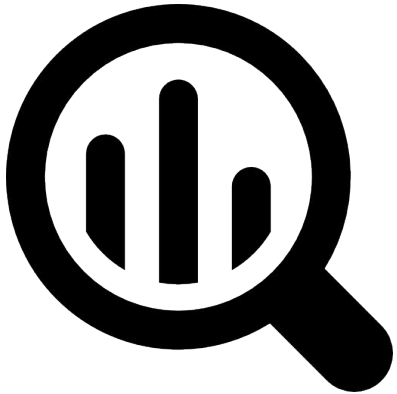

10. DFService Example

# Experimental Results

In the collection of relevant data for the experiments that follow, the **Analysis** class played a fundamental role.

Instantiated by JADELauncher, accessible by all Lift agents, this class allows us to store data(in a CSV file) such as:

Average Lift **occupation**
**Requests attended** by each lift(divided by different types of Request)
**Entrances** and **exits** on each floor



11. Data Gathering Icon

Each **Lift Agent's** also responsible for creating a csv file that stores the Lift's floor in a given tick.

# Experimental Results

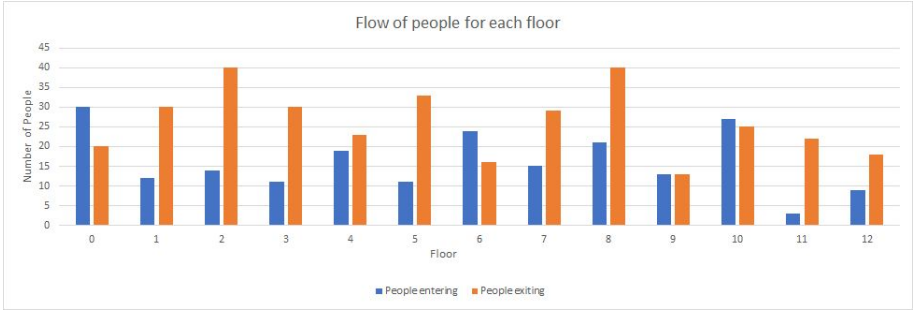
**Experiment 1**

| Number of Floors | Number of Lifts | Max Weight | Max Speed | Floor Distance |
|---|---|---|---|---|
| 12 | 2 | 600 | 2.5 | 5 |

**Key Data:**

1) Floor Analysis
2) Lift Tasks Analysis
3) Lift Pathing Analysis

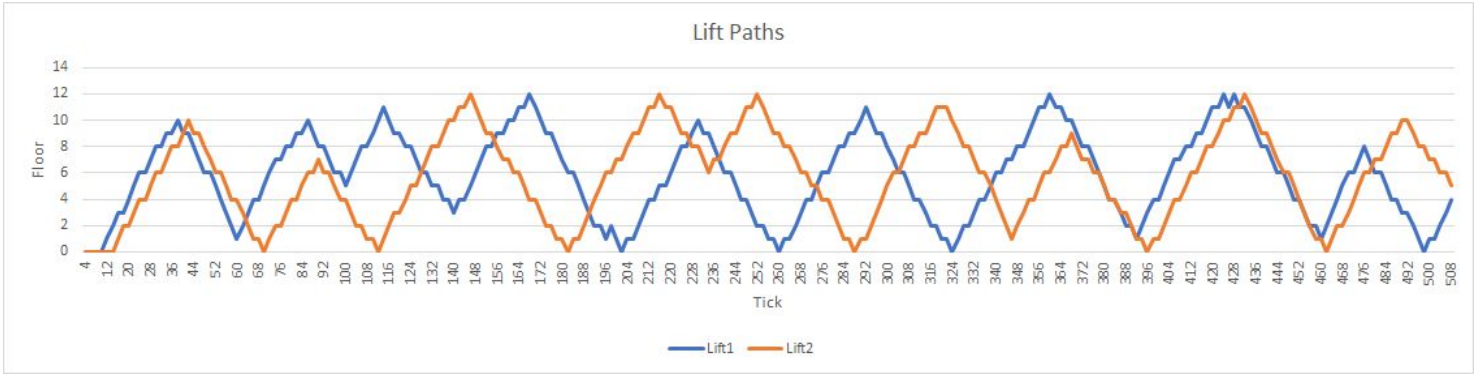
1)




2)

| Lift | n Up Requests | n Down Requests | n End Requests | Average Occupation % |
|---|---|---|---|---|
| 1 | 23 | 25 | 83 | 27% |
| 2 | 17 | 36 | 83 | 35% |

3)



# Experimental Results

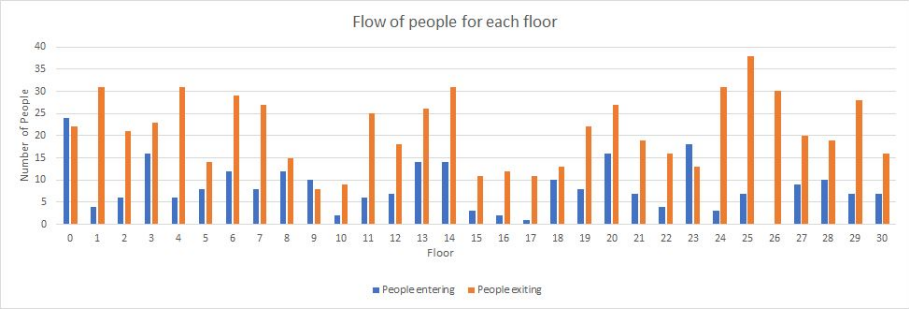
**Experiment 2**

| Number of Floors | Number of Lifts | Max Weight | Max Speed | Floor Distance |
|---|---|---|---|---|
| 30 | 6 | 600 | 2.5 | 5 |

**Key Data:**

1) Floor Analysis
2) Lift Tasks Analysis
3) Lift Pathing Analysis

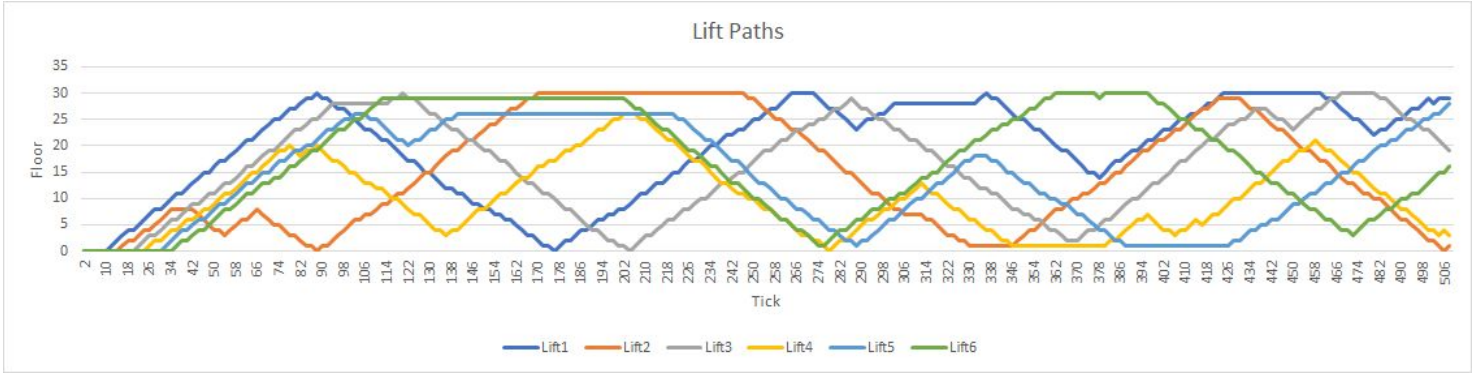
1)




2)

| Lift | n Up Requests | n Down Requests | n End Requests | Average Occupation % |
|---|---|---|---|---|
| 1 | 12 | 7 | 45 | 25% |
| 2 | 8 | 6 | 27 | 18% |
| 3 | 8 | 11 | 47 | 19% |
| 4 | 7 | 12 | 40 | 15% |
| 5 | 7 | 5 | 39 | 14% |
| 6 | 11 | 5 | 38 | 21% |

3)



# Conclusions

Besides the fact that our Lift Management System works properly, we found some features we could add and some points we could improve.

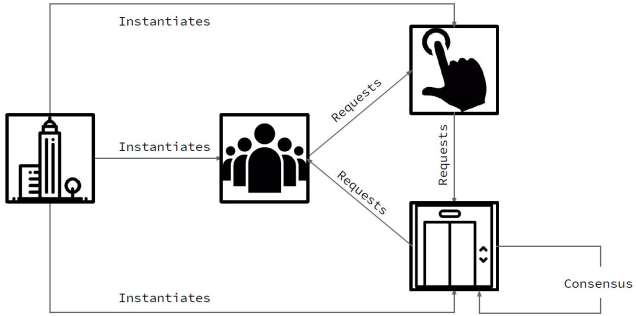
They range from:

- Enhancing our taskList insertion **Heuristic**
- Experimenting with other **Lift Request Strategies**
- Improving the **User Interface**

Having this said, we believe we have made a good job developing this **Multi-Agent System**. It is efficient and involves a lot of materials approached in our **AIAD** classes.


| **4MIEIC01** | **Group 13** |
| --- | --- |
| António Dantas | up201703878 |
| João Macedo | up201704464 |
| Vítor Gonçalves | up201703917 |

# Additional Information - Interaction Overview




The **Multi-agent System** is launched through the **JADELauncher** file, which is responsible for creating an instance of the **Building Agent** which is then responsible for creating all the other agents involved.

The **Building Agent** receives information regarding the total number of floors and lifts, as well as their characteristics (capacity, speed). The Request Agent, responsible for all requests that are made, is also **instantiated** by the Building.

The Request Agent **periodically generates new orders** to the FloorPanel Agent that simulates elevator calls on the floors, both Up and Down. FloorPanel forwards these messages to all Lifts who decide which one will be responsible for fulfilling the request (next slides).

When an elevator reaches the Floor where it has to fulfill a request, it asks the Request Agent for the extra information it needs, that is, people entering and leaving, and the destination floors.

# Additional Information - Request Processing

For a good understanding of the codebase it's important to follow the lift agent's **processing of a request** coming from a Floor Panel Agent.

1. Receiving request
2. **Calculating optimal task list position for request**
3. Calculating time to attend the request
4. Reaching consensus with a version of the Bully algorithm (Least time wins, ties resolved by Lift ID)
5. If allocated, update task list

We will now focus on point 2, as it lacks explanation on the rest of the presentation

The optimal position on the list is determined by the direction of movement. While going up the lift attempts to get all “UP” and “End” Requests, and while going down all “Down” and “End” Requests, if the floor is on it’s path. Additionally, turning points are also detected to ensure that, if need be, a lift can keep going in a particular direction to attend to a request, inserting the new task list entry on the turning point, when optimal.

# Additional Information - Request Processing

### Calculating optimal Position (HandleRequest.java line 62)

```
//Gets the list position for a new request
public int getListPos(LiftTaskListEntry entry) {

    if (myAgent.getTaskList().size() == 0) return 0;

    int lastPos = myAgent.getFloor();
    int pos = 0;

    switch (entry.getType()) {
        case End:
            for(LiftTaskListEntry taskListEntry : myAgent.getTaskList()) {
                if(between(lastPos, taskListEntry.getFloor(), entry.getFloor())
                        || between(taskListEntry.getFloor(), lastPos, entry.getFloor())
                        || shouldPlace(pos, entry))
                    return pos;
                lastPos = taskListEntry.getFloor();
                pos++;
            }
            return pos;
        case Down:
            for(LiftTaskListEntry taskListEntry : myAgent.getTaskList()) {
                if(between(taskListEntry.getFloor(), lastPos, entry.getFloor())
                        || shouldPlace(pos, entry))
                    return pos;
                lastPos = taskListEntry.getFloor();
                pos++;
            }
            return pos;
        case Up:
            for(LiftTaskListEntry taskListEntry : myAgent.getTaskList()) {
                if(between(lastPos, taskListEntry.getFloor(), entry.getFloor())
                        || shouldPlace(pos, entry))
                    return pos;
                lastPos = taskListEntry.getFloor();
                pos++;
            }
            return pos;
        default:
            return myAgent.getTaskList().size();
    }
}
```

### Should place on Turning Point (HandleRequest.java line 109)

```
//Returns true if should place Proposed Task for catching turning points
private boolean shouldPlace(int i, LiftTaskListEntry entry) {
    var tasks = myAgent.getTaskList();
    if (i > 2) {
        if (turningPoint(i)) {
            if (entry.getFloor() > tasks.get(i).getFloor()
                    && entry.getFloor() > tasks.get(i-1).getFloor()
                    && entry.getType() != Type.Up)
                return true;

            if (entry.getFloor() < tasks.get(i).getFloor()
                    && entry.getFloor() < tasks.get(i-1).getFloor()
                    && entry.getType() != Type.Down)
                return true;
        }
    }
    return false;
}
```

### Detecting Turning Point (HandleRequest.java line 129)

```
//Returns true if is turning point
private boolean turningPoint(int i) {

    var tasks = myAgent.getTaskList();

    if ( i < 2 ) return false;

    boolean goingUp = tasks.get(i-2).getFloor() < tasks.get(i-1).getFloor();
    boolean goingDown = tasks.get(i-2).getFloor() > tasks.get(i-1).getFloor();

    if(goingUp && tasks.get(i).getType() == Type.Down) return true;
    if(goingDown && tasks.get(i).getType() == Type.Up) return true;

    if(tasks.size() > i+1) {
        if(goingUp && tasks.get(i).getType() == Type.End)
            if(tasks.get(i).getFloor() > tasks.get(i+1).getFloor()) return true;
        if(goingDown && tasks.get(i).getType() == Type.End)
            if(tasks.get(i).getFloor() < tasks.get(i+1).getFloor()) return true;
    }

    return false;
}
```

# Additional Information - Lift Pathing

From the moment a request is issued by a person through the Floor Panel Agent, all Lift Agents are messaged.
Each lift must take into account its internal state, that is, its working queue. For this effect:

If a lift has reached its maximum capacity it **must not let** anyone else enter until it has dropped someone.
If the **call on floor X** indicates the person would like to **go to a floor greater than X** then all tasks on the queue for floors **lower than X should be completed** before attending the new request.
If there are **no active requests**, attending the number of lifts and floors in the building, the lifts must wait in their current floor until a request is placed.

# Additional Information - Ticker Behaviour

The Lift Agent's ticker behaviour is the most important behavior for the underlying function of the elevator system, the transportation of people. It assures, in the following order, that:

1. If the Lift is on the floor equal to the next task, it **processes the task** (slide: Additional Information - Fulfillment of a Task)
2. **Update real position** of lift (given by (tick_period / 1000) / lift_speed)
3. **Update lift Floor** from lift's real position
4. Recording position of lift for analysis

# Additional Information - Fulfillment of a Task

Each Lift Agent has a taskList, which allows him to know the order of the floors he has to attend.As soon as the current Floor of a Lift is the same as the Floor associated with the 1st element of his TaskList it has to fulfill the request.

For this, the elevator will send a message to RequestAgent with the current request it is fulfilling. After receiving the request, RequestAgent will generate a message with a specific structure containing all the information that the Lift will have to process to satisfy the entry/exit of people. It should be noted that in the UP and Down request types, the entry of people will always occur, with the departure being optional. In END type request(when the Lift stops for people to leave), the departure of people is mandatory.

When people enter, new destination floors can be associated, for this, RequestAgent generates a list of new floors that the elevator will have to stop to leave the new passengers.

**Examples following the message format created for this purpose:**

**E:3[1-2],S:2** => 3 people enter and intend to go to Floor 1 and 2. 2 people leave

**E:1,S:1** => 1 person enters whose destination is already served by the elevator. 1 person leaves

# Additional Information - Consensus Algorithm

We chose to use a **Bully Election Algorithm** to decide which Lift will attend the Request.

We implemented this as a **Behavior** that is initialized upon the Lift's Setup. When a request is received, a Lift will send its estimated time to all other Lift's and receive the other one's.

It will then compare its own proposed time with the rest of them. If it finds one that will get there quicker it will stop the comparison. Otherwise, if it is better than everyone else it will send an **HALT** message at the end.

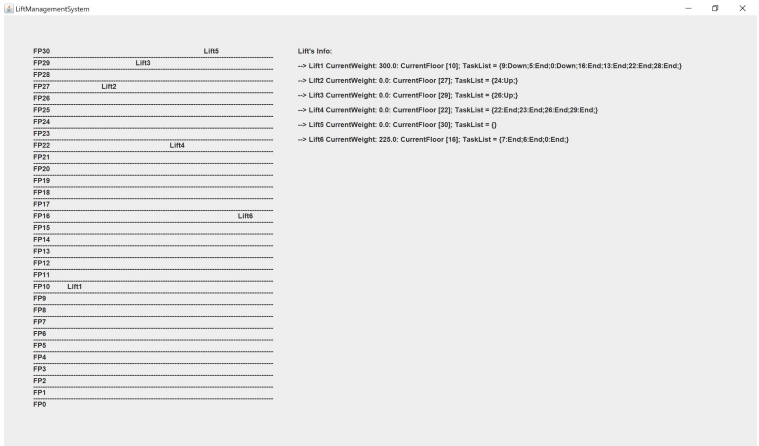
The other Lifts will then receive the **HALT** message and send their agreement to the elected Lift, which will add the request to its **taskList**.

# Additional Information - GUI using Swing

For the Graphical User Interface we used **Swing**.

We have got a class called **SwingDisplay** that draws the Building's State and each Lift's tasklist. This helps us to clearly see the Lift's movement and their trajectory.

The Screen is updated every 0.25 seconds using a **ScheduledExecutorService**.




12. Swing GUI