UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA E ELETRÔNICA

EEL7123 - TÓPICO AVANÇADO EM SISTEMAS DIGITAIS

RELATÓRIO FINAL

ELEVADOR

PROJETO FINAL

Autores:

Bruno Neckel Wesling

Pedro Henrique Kappler Fornari

Florianópolis, 30 de Novembro de 2015.

SUMÁRIO

1.INTRODUÇÃO.

Sabemos que até meados do século XIX quase não havia construções com mais de três pavimentos. Com o acelerado desenvolvimento urbano, e o espaço físico cada vez mais escasso, a demanda por uma solução vertical chamou a atenção. Nos dias atuais, o que pode ser visto é que o elevador está presente no cotidiano de todos, seja em casa, trabalho, ou nas lojas e supermercados. A vida moderna e vertical como conhecemos hoje não existiria sem este grande mas, por vezes, imperceptível advento da engenharia.

Hoje, com os avanços da tecnologia e da computação, várias ferramentas foram desenvolvidas com o intuito de modelar de forma eficiente e precisa os sistemas automatizados de controle de elevadores. O controle automático de um elevador tem como finalidade garantir o deslocamento seguro, rápido e eficiente, seja para passageiros ou para cargas.

Com o passar do tempo, grandes evoluções na área de eletrônica e instrumentação possibilitaram sensores cada vez melhores e confiáveis, e com isso, os projetos na área de tecnologia embarcada e automação foram se tornaram cada vez mais complexos. Com isso, as técnicas de projeto também precisaram sofrer modificações para que pudessem alcançar o grau de sofisticação e complexidade demandado. Dessa necessidade de mudança de técnicas de projeto é que surgiu a motivação para esse trabalho, como visto durante as aulas as técnicas de projeto vem sendo atualizadas buscando uma otimização no processo de desenvolvimento de novos produtos, rapidamente e com eficiência.

Este trabalho tem como objetivo aplicar os conceitos de etapas de projeto, analise de requisitos e modelagem do sistema, desenvolvimento e testes/verificação, na modelagem em software de um sistema de dois elevadores utilizando-se da linguagem C++.

Primeiramente serão abordados os requisitos do sistemas e a modelagem do mesmo fazendo-se uso de métodos de diagramas como: Use Case, Class Diagram, Sequence, Statecharts, FSM. Depois será apresentado o desenvolvimento de um

código em C++ e a etapa de testes e verificação do sistema desenvolvido. Por fim, será feito o fechamento do trabalho e as devidas conclusões serão expostas.

2.ANALISE DE REQUISITOS E MODELAGEM DO SISTEMA

O projeto iniciou-se pela análise dos requisitos fornecidos para o projeto e a partir deles iniciou-se a busca por ferramentas para desenvolvimentos dos diagramas necessários. Para desenho do diagramas foi utilizado o software yEd Graph Editor da yWorks.

2.1 USE CASE

Neste diagrama é representado a relação entre os usuários e o sistema, também é possível identificar os diferentes tipos de usuário.

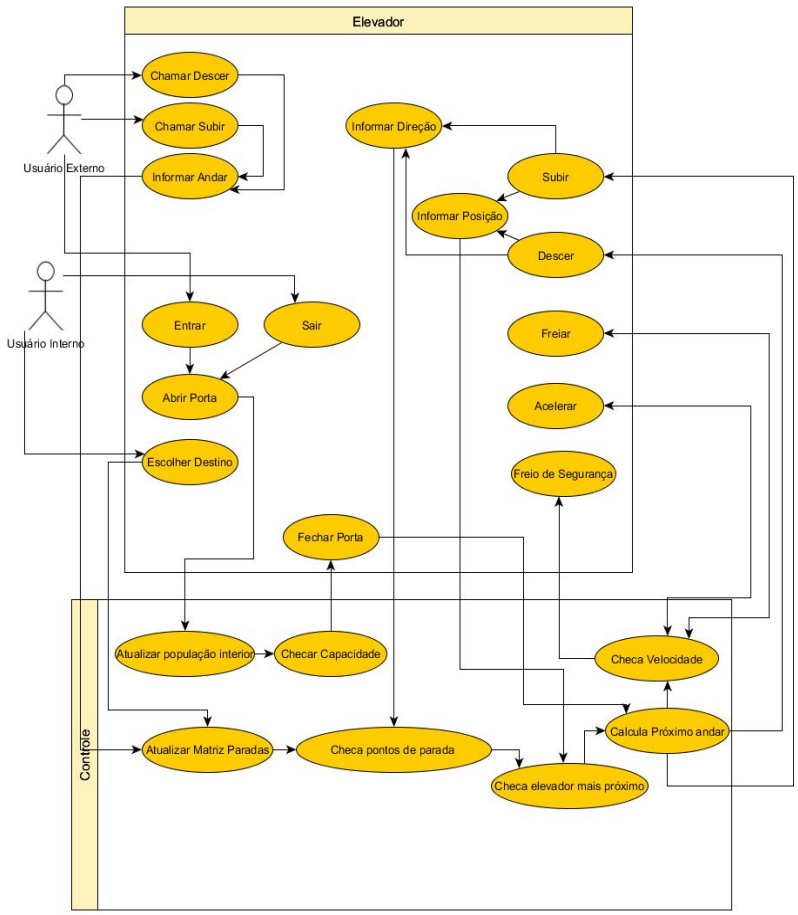

Figura 2.1:Use Case.

2.2 CLASS DIAGRAM

Representação da estrutura e relação entre as diferentes classes tendo-se assim todas as classes que o sistema necessita.

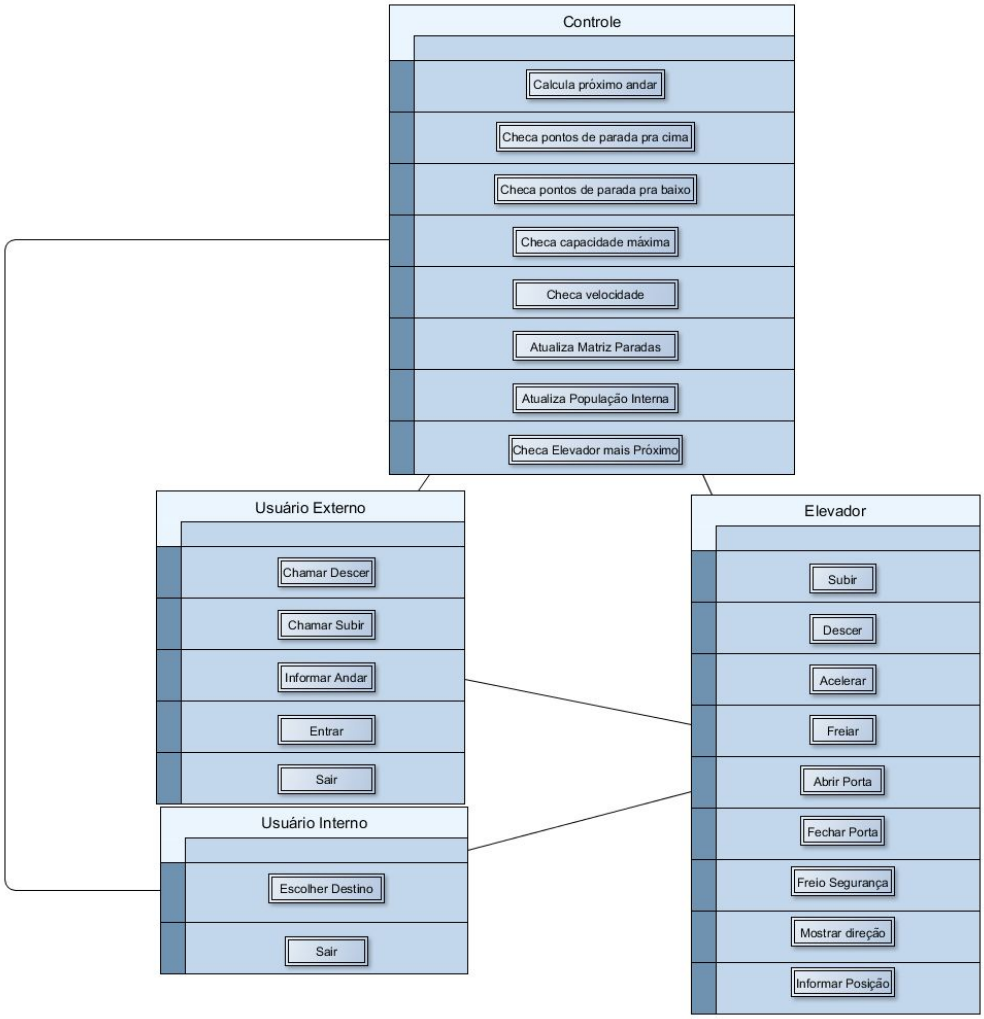

Figura 2.2: Class Diagram.

2.3 SEQUENCE

Este diagrama demostra como uma operação interage com as outras e qual a ordem de operação. As regiões quadriculadas demostram os laços do sistema.

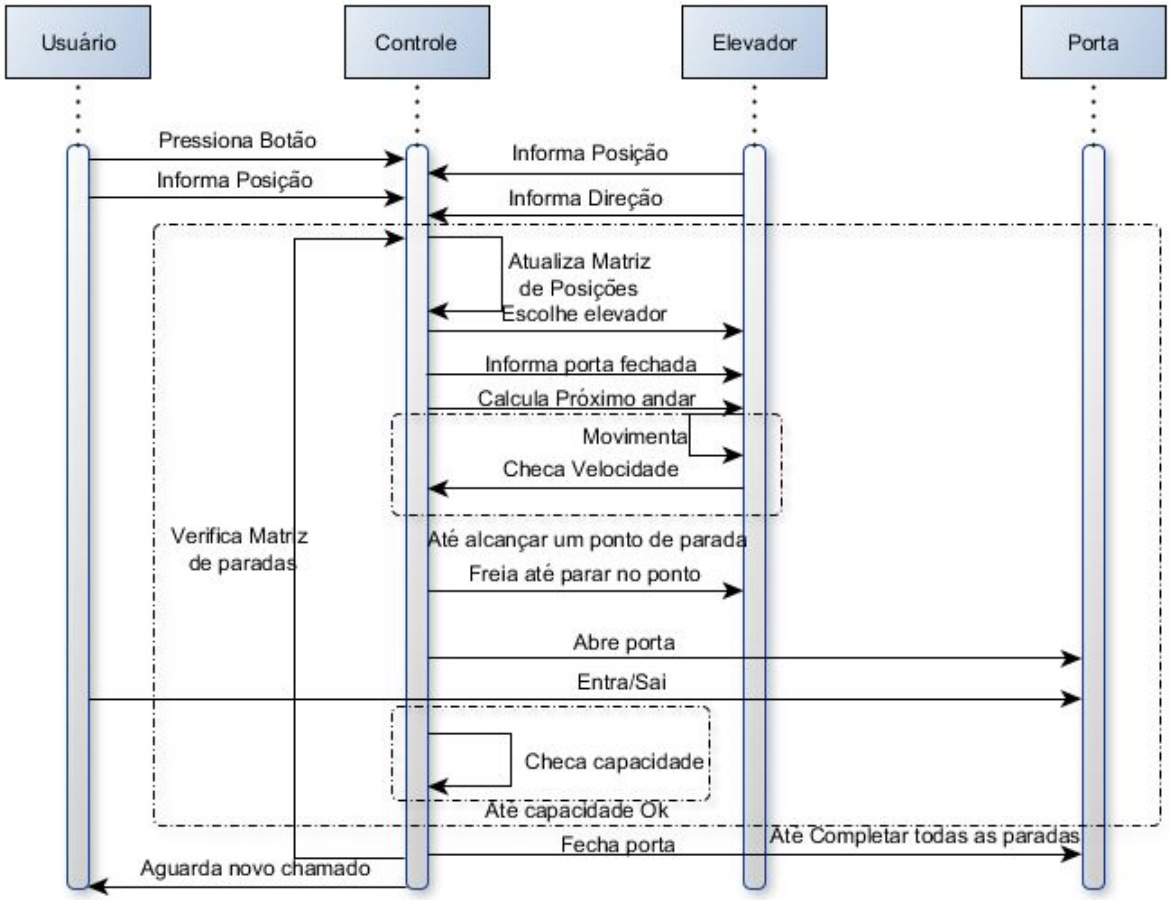

Figura 2.3: Diagrama do tipo Sequence.

2.4 STATECHARTS

Os diagramas abaixo ilustram a sequencia de estados que o sistema segue. Primeiramente temos um visão geral e depois os respectivos detalhamentos. Inicialmente temos um estado de espera aonde é aguardado alguma chamada, assim que algum usuário fizer uma solicitação as informações são coletadas e a partir delas é executada alguma das funções de escolher o elevador, movê-lo ou interagir com a porta. Em paralelo temos o estado de segurança que pode ser ativado a qualquer instante caso a velocidade seja excedida. As linhas com flechas significam uma entrada no estado enquanto que as outras representam uma saída.

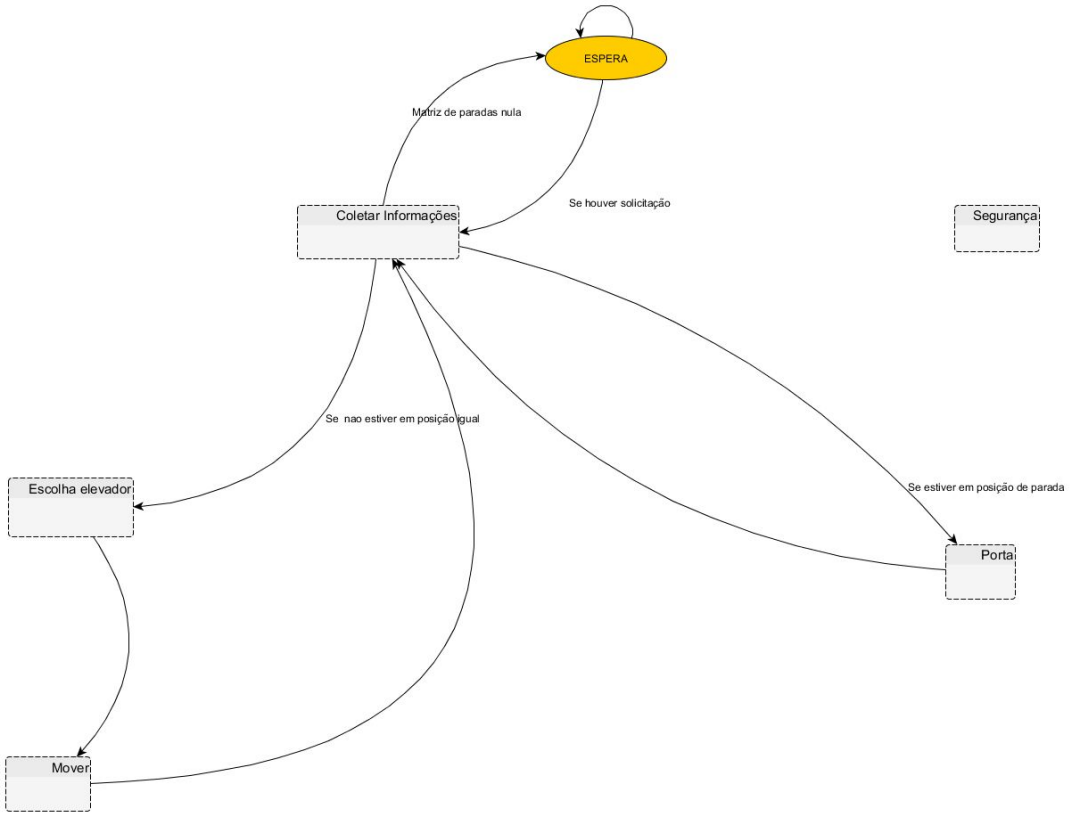

Figura 2.4.1: Statechart geral.

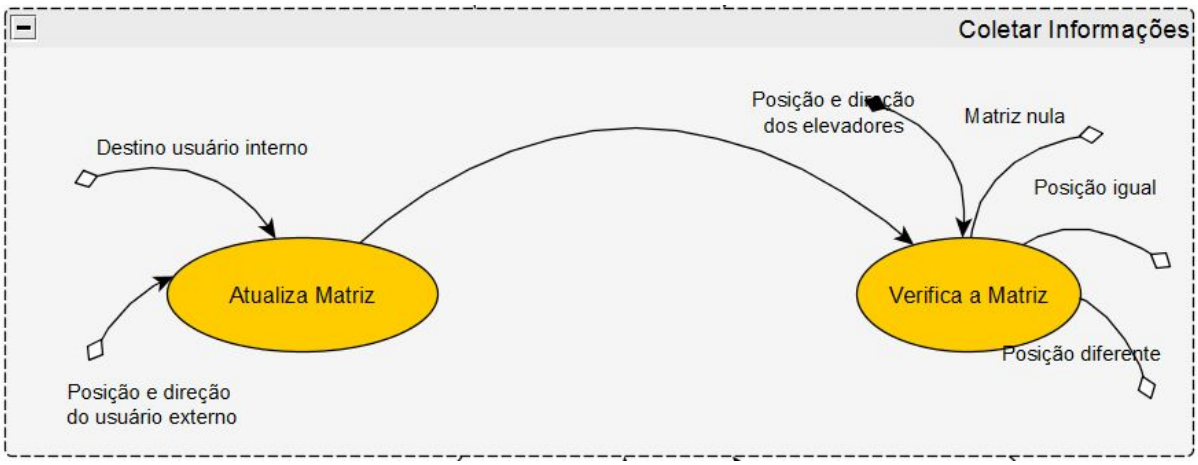

Figura 2.4.2: Statechart Coletar Informações detalhado.

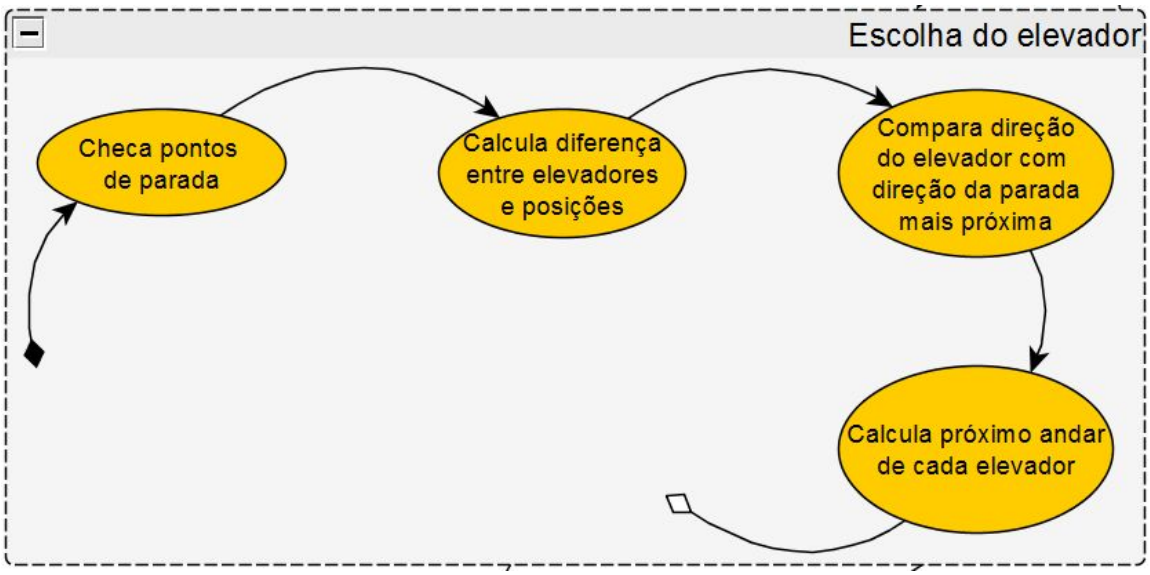

Figura 2.4.3: Statechart Escolha do elevador detalhado.

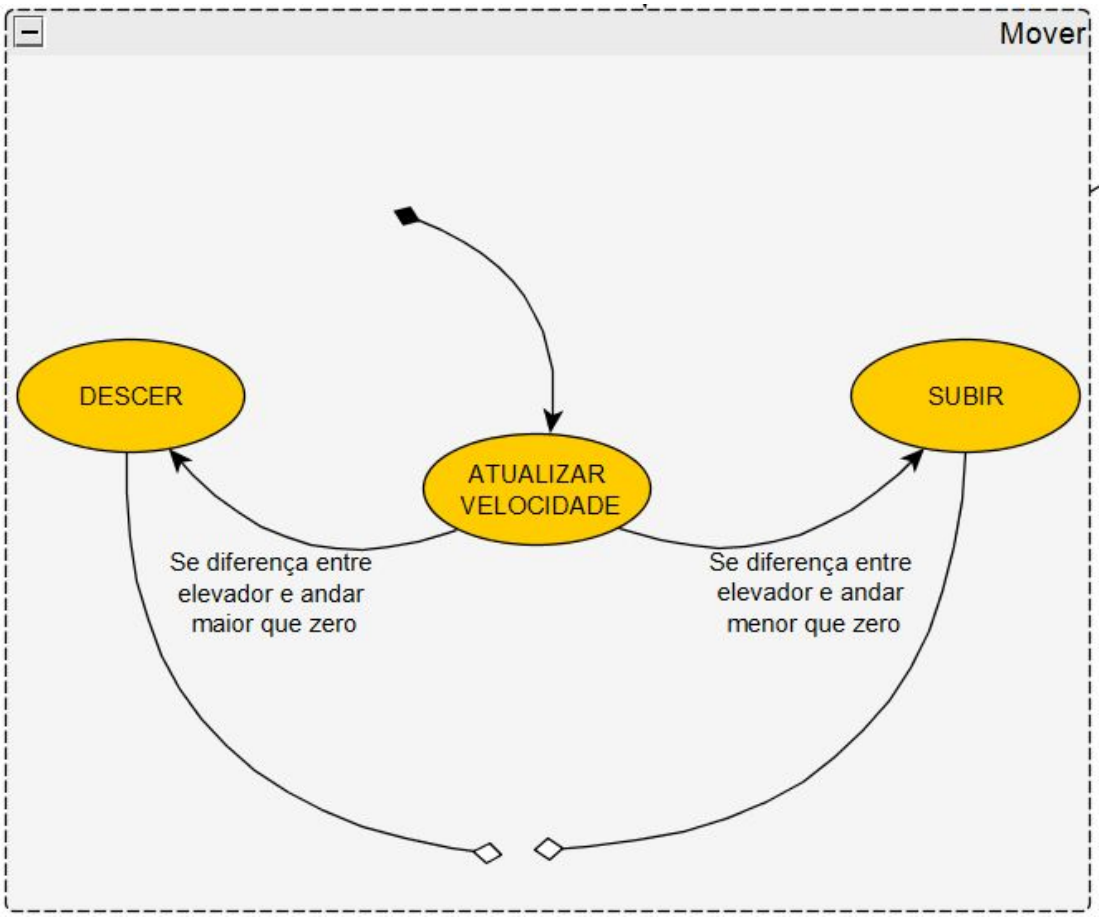

Figura 2.4.4: Statechart Mover detalhado.

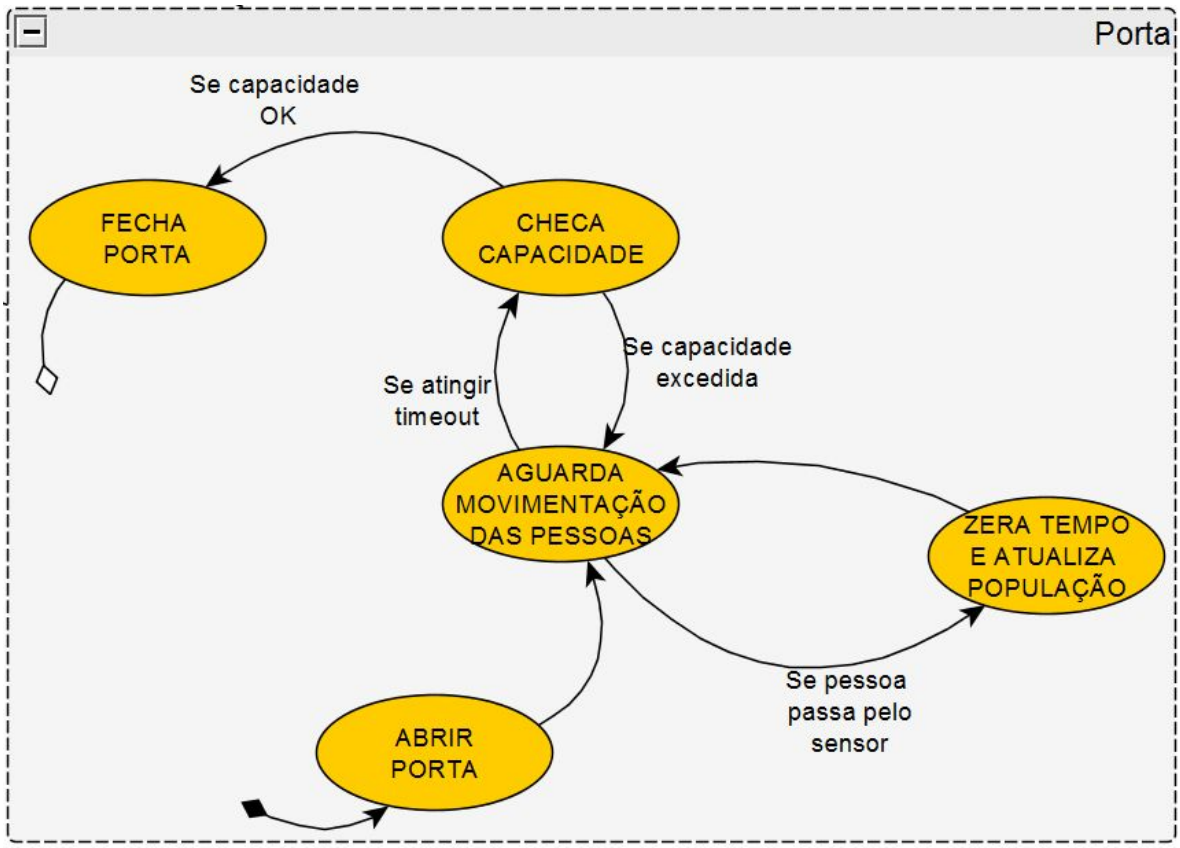

Figura 2.4.5: Statechart Porta detalhado.

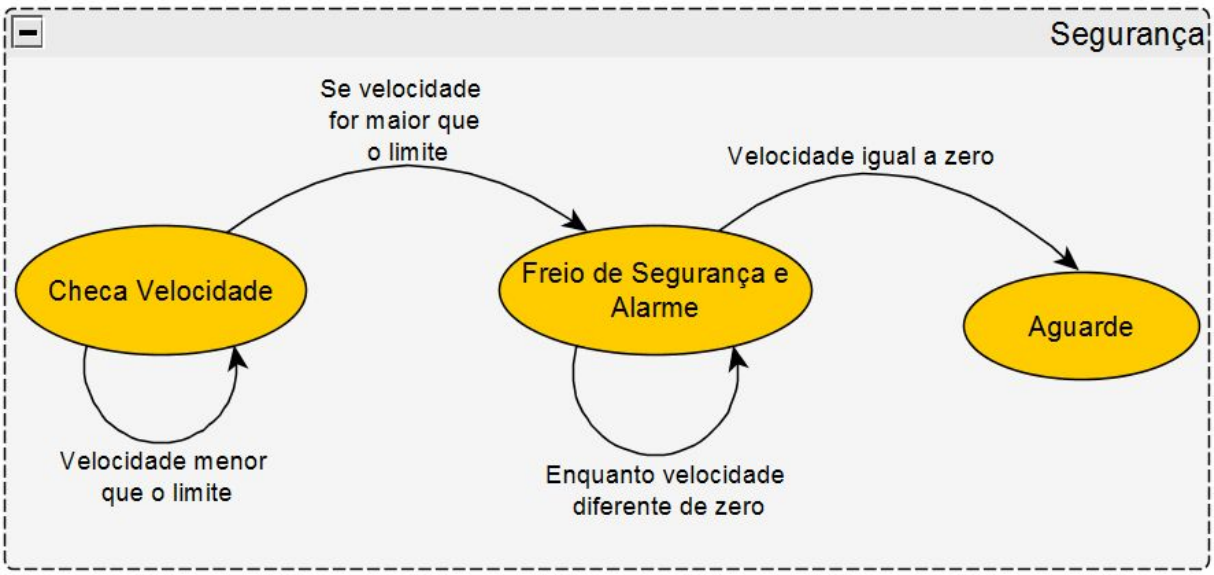

Figura 2.4.6: Statechart Segurança detalhado.

2.5 FSM

Máquina de estados do sistema desenvolvido.

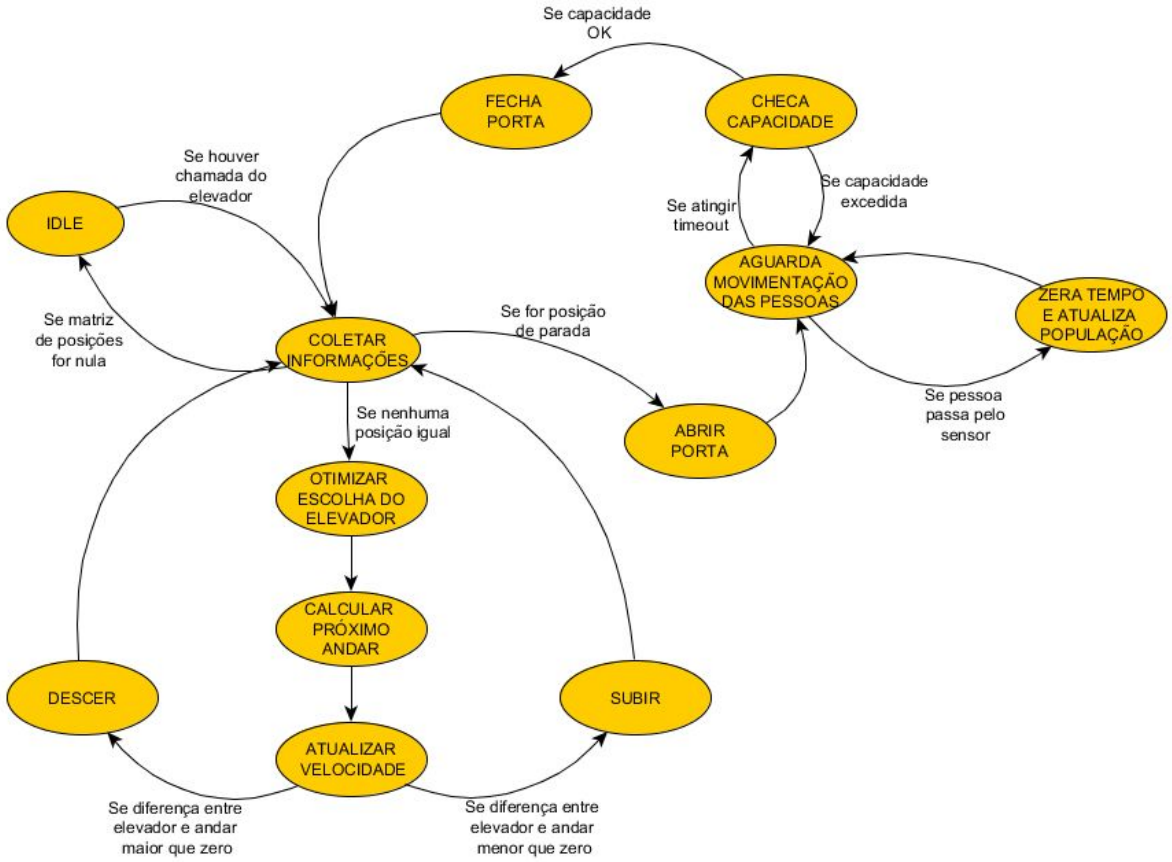

Figura 2.5: FSM geral do elevador.

3.DESENVOLVIMENTO

Nesta etapa do projeto, desenvolvemos três arquivos, um header, descrevendo as classes, variáveis públicas e privadas de cada uma e protótipos das funções; um arquivo para implementar as classes com suas devidas funções e por fim um arquivo com a implementação de uma simulação do projeto, onde fizemos testes e verificações para detecção e correção de bugs em cada função desenvolvida.

3.1.ELEVADOR.H

Este arquivo foi onde criamos as classes, contendo o protótipo de cada método a ser implementado, definicões de constantes utilizadas para a implementação dos métodos, além da relação criada entre classes, fizemos com que as classes elevador, internal_user e external_user tivessem herança da classe control, para que a classe control pudesse ter acesso ao restante as classes e realmente controlar o sistema.

Além disso as declarações de variáveis de cada classe também foi definida neste arquivo, utilizando inicialmente todas como variáveis privadas da cada classe, porém para facilitar a simulação optamos por deixar algumas delas como públicas, de forma que poderiam ser acessadas por qualquer arquivo.

3.2.ELEVATOR.CPP

Neste arquivo encontram-se as implementações de todos s métodos do sistema, separados por classe. foram feitos também comentários para explicar cada função, porém iremos detalhar um pouco de cada rapidamente.

A classe elevador, como visto na Figura 2.2 possui métodos para movimentação para cima e para baixo, para abrir e fechar portas, para contabilizar a entrada e saída de passageiros, para controlar a velocidade além de uma função que retorna seu respectivo andar, velocidade ou quantas pessoas estão dentro do elevador. Além disso foram criadas 3 matrizes de dados para cada elevador, onde a matriz_paradas informa quais são as paradas que o elevador deverá realizar, além disso essa matriz contém informações sobre condições de parada, como, por exemplo, parar indefinidamente, parar subindo ou descendo, etc. Com essa matriz e alguns métodos da classe control, as matrizes paradas_cima e paradas_baixo são atualizadas, de forma que dependendo do estado do elevador o próximo andar será calculado, porém isso será melhor explicado na classe control.

As classes internal_user e external_user fazem funções dos usuários, como chamar elevador, escolher andar de destino, entre outros, porém separamos as duas para ter certeza de que sempre estaremos trabalhando com o real estado do

usuário. Dessa forma, o método de chamar elevador é utilizado apenas pelo usuário externo, que informa seu andar e se deseja subir ou descer, já o método que seta o andar de destino é utilizado apenas pelo usuário interno, o qual informa em qual elevador se encontra e qual andar deseja ir.

Por fim, a classe control, como o nome já sugere, controla um sistema de dois elevadores, tentando otimizar o tempo e a energia gasta pelos elevadores. Para isso, como apresentamos nas Figuras 2.4.1 e 2.5, o elevador movimentasse um andar por vez e, a cada repetição, é feito um novo cálculo para verificar quais são as paradas necessárias e qual delas tem maior prioridade. Esse cáculo é feito verificando qual é a condição de parada e preenchendo as matrizes paradas cima e paradas baixo de cada elevador, com isso o elevador, que pode possuir três estados, parado, subindo ou descendo, verifica as posições de parada com relação ao seu estado atual e as compara com seu andar atual, escolhendo a menor distância. É então calculado qual será o próximo andar do elevador, assim como sua próxima velocidade, criando um movimento contínuo.

Essa classe também checa a capacidade de cada elevador e a velocidade de cada elevador, sendo que inicia a frenagem quando a distância entre um andar de parada e o andar do elevador for menor que a velocidade atual, também é possível acionar um freio de segurança caso a velocidade do elevador for excedida, porém a função não foi testada. Além disso as portas não se fecham enquanto o elevador estiver com a capacidade excedida, como apresentado nos diagramas.

3.3.MAIN.CPP

Este arquivo serviu como um simulador, onde, com o tempo, introduzimos alguns métodos e testamos separadamente cada bloco da máquina de estados apresentada nas Figuras 2.4.1 até 2.5 com isso realizamos alguns testes, que serão apresentados no próximo tópico, porém que serviram para corrigir alguns erros que estávamos encontrando.

Atualmente o arquivo simula o movimento de 3 usuários pelo sistema, onde o primeiro encontra-se no térreo e os demais em andares distintos, distribuídos pelo elevador. quando executado o programa apresenta os andares em que cada

elevador se encontra, assim como sua velocidade atual. Assim que o último usuário deixa o elevador a simulação é encerrada.

4.TESTES/VERIFICAÇÃO

Após corrigir todos os erros de sintaxe apresentados pelo compilador, testamos o comportamento do elevador quando um usuário externo utilizava a função para chamar o elevador, tanto para subir quanto para descer. Isso nos fez identificar alguns erros de sintaxe mais avançados, como a utilização de sequências de if's sem um compromisso em abordar apenas uma condição. Feito algumas correções na implementação dos métodos o sistema funcionava bem para um usuário movimentando-se em uma única direção.

Adicionamos então um segundo usuário e verificamos o comportamento do sistema, com ajuda da função cout, conseguimos verificar erros mais detalhistas, porém, que estavam causando um grande impacto no funcionamento do sistema, como uma continuação sem parar em andares desejados ou até mesmo ficar travado no primeiro usuário.

Utilizar o cout foi a opção que encontramos para substituir o GDB, que estudamos em sala de aula, porém que estava com problemas para debugar códigos em C++, utilizar o GDB facilitaria bastante, pois poderíamos acompanhar o código com break points e no modo step by step, o que facilitaria entender onde o programa se perdia.

Assim fizemos sucessivamente, até chegarmos ao programa que anexamos neste trabalho, porém com esse sistema de detecção ainda conseguimos identificar alguns bugs que não fomos capazes de corrigir a tempo da entrega do trabalho, como, por exemplo o controle de velocidade, que em alguns momentos se mantem em 0m/s e na realidade está se movimentando, o que ocorre por causa da maneira em que implementamos as funções de controle de velocidade, onde quando a diferença entre andares desejados fosse maior que a velocidade haveria uma frenagem, porém quando o elevador para em um andar e terá que parar dois andares após aquele ele irá acelerar e frear a cada andar, o que seria ruim na prática.

Outro bug identificado foi que quando um elevador está com a porta aberta recebendo um passageiro, o outro não abre as portas, o que ocorre pela maneira em que implementamos a sequência de if's para verificar quando o elevador chegou em um andar em que deve parar, e seria corrigido apenas alterando bastante a maneira como foi desenvolvido o simulador. Ainda existe outro bug, no qual, quando existem dois usuários no mesmo andar, e o elevador para neste andar o simulador só permite a entrada de um usuário e exclui aquela posição da matriz de paradas, de forma que o outro usuário teria que chamar o elevador outra vez. Além destes erros, outros podem existirem, porém seria necessário mais tempo de verificação e correção para deixarmos o sistema ótimo tanto na simulação quanto na prática.

5.CONCLUSÃO

Com este trabalho foi possível verificar que entre as vantagens da utilização de uma metodologia de desenvolvimento de projeto pode-se destacar que a partir do momento em que concluímos a etapa de modelagem do sistema fazendo-se uso de diagramas, a etapa de desenvolvimento do sistema se torna mais rápida e eficiente, permitindo assim que erros de projeto possam ser detectados antes mesmo de ser posto em operação de teste e evitando re-spin.

Além disso, alguns tópicos dentro deste projeto em específico foram bastante interessantes e desenvolveram nosso conhecimento com C++ e com a própria lógica de algorítimos de otimização, porém alguns pontos, como a utilização de interrupções para sistemas de botões para os usuários, assim como para a ativação dos freios de segurança seriam bem mais eficientes, porém não tivemos ferramentas e conhecimento suficiente neste projeto.

Também é possível verificar que mesmo com um elevador funcional, ainda existem várias melhorias que podem ser feitas, deixando o sistema melhor otimizado, tanto em energia quanto em velocidade de resposta para os usuários, o que pode ser feito com uma verificação mais rígida e com um tempo de desenvolvimento um pouco maior.

Ainda podem ser adicionados mais subsistemas como alarmes, sinalizações para o usuário estar a par do que está acontecendo e a utilização de algoritmos com

rotinas mais inteligentes para avaliar o tráfego de passageiros e selecionar a lógica de atendimento mais apropriada. As etapas de projeto utilizadas possuem aplicações praticamente infinitas, e este trabalho apesar de direcionado a controle de elevadores, pode ser utilizado para outras aplicações na área de sistemas embarcados.