DESENVOLVIMENTO DE UM SIMULADOR DE TRÁFEGO PARA COMPARAÇÃO ENTRE PARADIGMA ORIENTADO A NOTIFICAÇÕES E PARADIGMA IMPERATIVO

Leonardo Faix Pordeus
leonardopordeus@alunos.utfpr.edu.br
Disciplina: **Programação Avançada** – Prof. Dr. Jean M. Simão
Programa de Pós Graduação em Engenharia Elétrica e Informática Industrial – CPGEI - Campus de Curitiba
Universidade Tecnológica Federal do Paraná - UTFPR
Avenida Sete de Setembro, 3165 - Curitiba/PR, Brasil - CEP 80230-901

Resumo – A disciplina de programação avançada exige o desenvolvimento de um software que contenha uma determinada quantidade de conceitos relacionados à orientação a objetos em C++. Para este trabalho escolheu-se o desenvolvimento de uma aplicação chamada CTA Simulator, na qual têm como objetivo simular o tráfego de uma região urbana. Os objetivos deste projeto são desenvolver estratégias de controle de semáforos, simular regiões de tráfego em uma área urbana, comparar o desempenho das estratégias, comparar o desempenho e complexidade quando as estratégias de controles forem implementadas em paradigmas diferentes. Para o desenvolvimento deste trabalho foram levantados os requisitos textualmente, com base no documento CONOPS. O projeto de *software* foi feito através de Diagrama de Classes e Diagrama de Estados em *Unified Modeling Language* (UML). Após o projeto de *software*, realizou-se o desenvolvimento em C++, no qual foram utilizados os conceitos aprendidos em sala de aula, como: coesão, desacoplamento, polimorfismo, classes abstratas, listas encadeadas, sobrecarga de operadores, herança, *Standard Template Library* (STL), threads, padrões de projetos, entre outros. Após a implementação foram realizados testes com o objetivo de verificar a sua funcionalidade conforme os requisitos levantados.

Palavras-chave: Simulação de controle de tráfego, Orientação a Objetos, Paradigma Orientado a Notificações, Comparações entre PON e IP

Abstract - The advanced programming course requires the development of software that contains a certain amount of concepts related to object oriented C++. For this work was chosen to develop an application called CTA Simulator, which aim to simulate the traffic of an urban area. The objectives of this project are to develop control strategies of traffic lights simulate traffic areas in an urban area, to compare the performance of strategies, compare the performance and develop complexity when control strategies are implemented in different paradigms. To develop this work, the requirements were made textually, based on the CONOPS document. The software project was done through Class Diagram and States Diagrams in Unified Modeling Language (UML). After the software project, there was development in C++, where we used the concepts learned in the classroom, such as cohesion, decoupling, polymorphism, abstract classes, linked lists, operator overloading, inheritance, Standard Template Library (STL), threads, design patterns, among others. After implementing tests were conducted in order to verify its functionality according to the raised requirements

Key-words: Traffic control simulation, Object Oriented, Notification Oriented Paradigm, Comparisons between NOP and IP

I. INTRODUÇÃO

O CTA Simulator é um projeto iniciado na disciplina Paradigma Orientado a Notificações do programa de pós-graduação em engenharia elétrica e informática industrial (CPGEI) da UTFPR e dada continuidade na disciplina de programação avançada do mesmo programa de pós-graduação.

Os objetivos deste projeto são desenvolver estratégias de controle de semáforos, simular regiões de tráfego em uma área urbana e apresentar a simulação sob a forma gráfica para o usuário. Uma primeira versão do CTA Simulator já havia sido desenvolvida na disciplina de programação avançada por outro aluno. A partir desta versão, na disciplina de paradigma orientado a notificações, foram desenvolvidas novas estratégias de controle sob este novo

paradigma, com a finalidade de comparação de desempenho e complexidade de desenvolvimento [1].

Neste trabalho é feita uma reengenharia do software CTA Simulator, fazendo a implementação de requisitos que não foram implementados nos trabalhos anteriores, analisando do projeto do ponto de vista de coesão e desacoplamento, aplicando novos conceitos de padrões de projeto, como o *Abstract Factory* [2] para criação de objetos, criação de uma interface gráfica e um segundo software para configuração da simulação. Iniciou-se pela análise dos requisitos através do documento CONOPS [3], que descreve os conceitos de operação do sistema. Após a análise dos requisitos, foi desenvolvido o diagrama de classes em UML e diagramas de estados com as lógicas das estratégias de controle dos semáforos. Com os diagramas prontos, iniciou-se o desenvolvimento do sistema utilizando a linguagem C++ em conjunto com a ferramenta de desenvolvimento Visual Studio 2013 Professional. Por último foram realizados testes através da visualização da interface gráfica desenvolvida, e através de arquivos de logs gerados pelo sistema.

As próximas seções estão organizadas da seguinte maneira: a Seção II descreve a aplicação CTA Simulator. Na Seção III é apresentado o desenvolvimento da aplicação. A Seção IV apresenta a tabela de conceitos da disciplina que foram utilizados. As conclusões são discutidas na Seção V.

II. EXPLICAÇÃO DO CTA SIMULATOR

Os objetivos do CTA Simulator são desenvolver estratégias de controle de semáforos, simular regiões de tráfego em uma área urbana, comparar o desempenho das estratégias, comparar o desempenho e complexidade quando as estratégias de controles forem implementadas em paradigmas diferentes, como o paradigma imperativo e o paradigma orientado a notificações [1].

O simulador representa objetos do mundo real, como veículos, ruas, pistas, quadras, sinaleiros, sensores e cruzamentos para uma determinada região de simulação matricial, composta por dez ruas verticais e dez ruas horizontais de mão única, formando uma matriz 10x10, com um total de 100 intersecções [3]. A figura 1 apresenta a região de simulação.

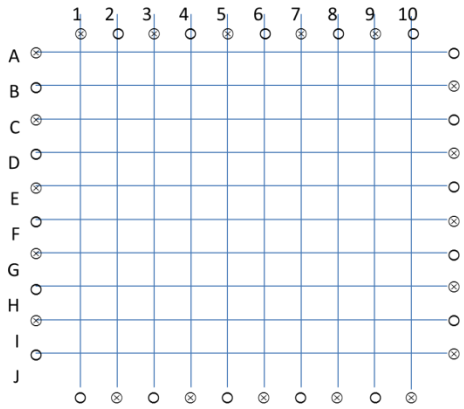

Figura 1. Região de Simulação [3].

Cada interseção da região simulada é composta por dois sinaleiros, um em cada rua, que têm os mesmos estados de um sinaleiro do mundo real: verde, amarelo e vermelho. A união de um sinaleiro para a rua vertical e outro para a rua horizontal é chamada de semáforo e, por questões de segurança, os sinaleiros não podem estar verdes simultaneamente [3].

Cada quadra possui o comprimento de 100 metros e capacidade de 25 veículos por pista, sendo que cada rua pode possuir de 1 a 4 pistas. Os veículos são criados de forma constante, no intervalo de 0.1, 0.2, 0.3, 0.4 ou 0.5 veículos por segundo. Quando um veículo é criado em uma entrada, ele começa a se movimentar a uma velocidade de 20m/s, e deve parar de se movimentar quando o sinal da quadra em que se encontra estiver vermelho ou no caso da próxima quadra estar cheia. Em cada cruzamento, uma porcentagem de veículos poderá virar para uma outra quadra. Esta porcentagem pode variar de 5% até 35% [3].

Em todas as quadras são instalados sensores que monitoram a quantidade de veículos que estão parados num sinal vermelho. Esses sensores apresentam três estados: FEW, MANY e FULL. Para o sensor estar no estado FEW a taxa de ocupação da rua deve ser menor do que 60%, para o estado MANY, a taxa de ocupação deve estar entre 60% e 99% e para o estado FULL, a taxa de ocupação deve ser 100% [3].

Para o simulador CTA, foram levantadas três alternativas de estratégia de controle de semáforos: controle independente, controle baseado em congestionamento e controle baseado em tráfego facilitado [3].

No controle independente, cada semáforo possui tempos fixos para cada estado do sinaleiro, não sendo considerados os sensores de quantidade de veículos e tempos de semáforos vizinhos.

Na estratégia de controle baseada em congestionamento é avaliado o tempo de cada semáforo e o estado referente à quantidade de veículos parados. Se o sensor detecta que a porcentagem de veículos parados está entre 60% até 100%, e o tempo do sinaleiro em vermelho é menor do que 24 segundos, então o tempo total do sinaleiro no estado vermelho é ajustado para 30 segundos. Caso o sensor detecte que a taxa de ocupação está entre 60% e 100% e o tempo do semáforo vermelho está entre 25 segundos e 39 segundos, o sinaleiro oposto altera imediatamente para o estado amarelo e o tempo restante do sinaleiro no estado vermelho é ajustado para 6 segundos. Se o sensor detectar que a ocupação está entre 60% e 100%, e o tempo do sinaleiro em vermelho for maior do que 39 segundos, não é realizada nenhuma alteração no tempo do sinaleiro.

No controle baseado em tráfego facilitado, para cada semáforo, um de seus sinaleiros possui uma *flag* sinalizando que a rua possui tráfego facilitado. Neste método de controle, o semáforo conhece o estado dos sensores de quantidade de tráfego e o tempo do semáforo anterior.

Neste caso, se o nível de congestionamento for menor do que 60% o atraso com relação ao semáforo anterior é de 5 segundos, ou seja, o sinaleiro abre 5 segundos após o sinaleiro do semáforo anterior abrir. Se o nível de congestionamento estiver entre 60% e 99% não há diferença no tempo de abertura com os semáforos anteriores. Se o nível de congestionamento for igual a 100% o atraso com relação ao semáforo anterior é de -5 segundos, ou seja, o semáforo abre 5 segundos antes do que o semáforo anterior [3].

III. DESENVOLVIMENTO DO CTA SIMULATOR

Nesta seção são apresentados os principais detalhes de análise e desenvolvimento do projeto. O CTA foi dividido em dois módulos, o simulador e uma aplicação externa para configuração de parâmetros a serem utilizados no decorrer da simulação.

A figura 2 apresenta os casos de uso do CTA Simulator, do ponto de vista do usuário. Para iniciar a simulação é gerado um arquivo de configuração através da aplicação CTAGenerator desenvolvida em conjunto com o projeto. Após ser gerado o arquivo, o usuário da aplicação CTA Simulator deve inserir este arquivo de configuração na mesma pasta onde se encontra o arquivo executável do CTA Simulator e iniciar a execução da aplicação. Ao ser iniciada a aplicação, esta carrega o arquivo de configuração, constrói a

região de simulação e começa a executar a simulação, mostrando seu resultado através de uma interface gráfica e através de um arquivo de log.

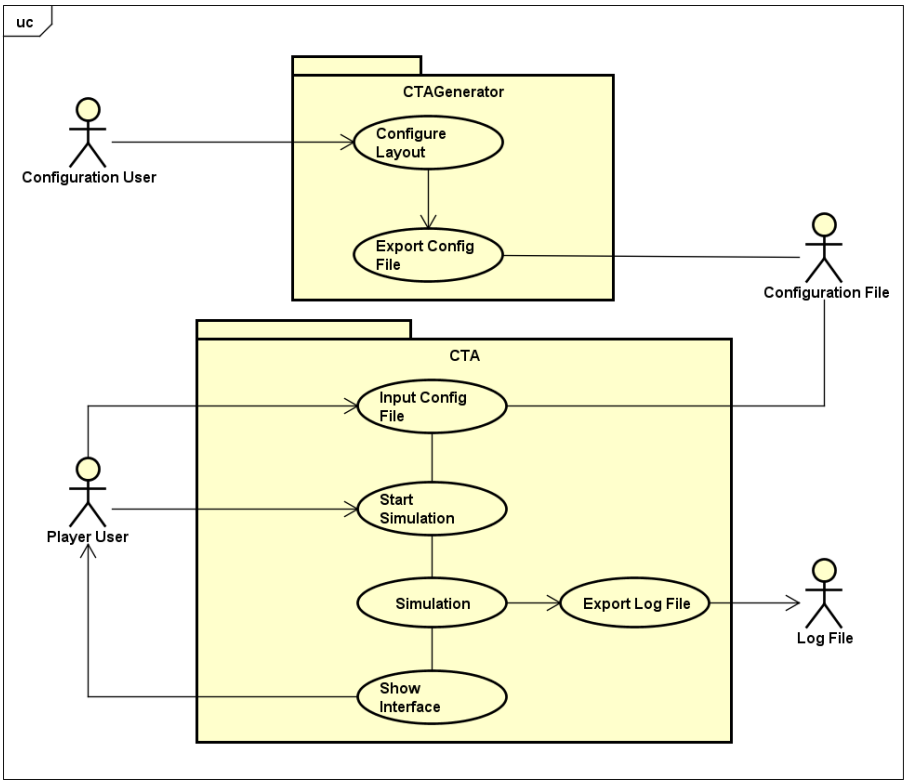

Figura 3. Casos de usos do CTA Simulator.

A Tabela 1 apresenta a lista de requisitos e suas situações com relação a sua implementação no projeto.

Tabela 1. Lista de Requisitos e suas Situações.

Requisitos Funcionais	**Situação**
O CTA deve simular os seguintes elementos: *Vehicle* *Lane* *Block* *Intersection* *Traffic Lights* *Semaphore*	Requisito previsto inicialmente e realizado
A região de simulação do CTA deve ser configurada a partir de um arquivo a ser importada pela aplicação.	Requisito previsto inicialmente e realizado
O CTA poderá configurar ruas com 1, 2, 3 ou 4 pistas	Requisito previsto inicialmente e realizado
Uma quadra ser configurada com o tamanho 100 metros.	Requisito previsto inicialmente e realizado
O CTA deverá permitir espaço para 25 veículos em cada pista	Requisito previsto inicialmente e realizado

de uma quadra.	
O CTA deverá permitir ruas de mão única, com direção N, S, E, W.	Requisito previsto inicialmente e realizado
O CTA deverá simular os estados de um sinaleiro real: Vermelho Amarelo Verde	Requisito previsto inicialmente e realizado
O CTA deverá simular a movimentação de um veículo. Um veículo está parado (devido a um sinal vermelho ou bloqueado pelo veículo da frente) ou se desloca uma velocidade de 20m/s.	Requisito previsto inicialmente e realizado
O CTA deverá simular uma intersecção entre duas ruas. Em cada intersecção existem 2 sinaleiros, um para cada rua do cruzamento. Estes dois semáforos devem estar sempre sincronizados.	Requisito previsto inicialmente e realizado
O CTA deverá simular a intensidade de tráfego de veículos. Os veículos são criados em pontos de entrada, de forma constante, através de um número no intervalo de 0.1, 0.2, 0.3, 0.4 ou 0.5 veículos por segundo.	Requisito previsto inicialmente e realizado
O CTA deverá simular sensores de intensidade de tráfego. Cada quadra possui sensores de veículo que monitoram a quantidade de veículos presentes. Quando o limite de 60% ou 100% é atingido, estes sensores notificam o controlador que então pode vir a mudar o estado dos semáforos afim de otimizar o tráfego.	Requisito previsto inicialmente e realizado
O CTA deverá fazer a simulação com base na estratégia de controle independente (IC).	Requisito previsto inicialmente e realizado
O CTA deverá fazer a simulação com base na estratégia de controle baseado em congestionamento (CBCL).	Requisito previsto inicialmente e realizado

O CTA deverá fazer a simulação com base na estratégia de controle baseado em tráfego facilitado (CBTF).	Requisito previsto inicialmente e parcialmente realizado
O CTA deverá permitir a visualizar a simulação a partir de uma Interface Gráfica.	Requisito previsto inicialmente e realizado
O CTA deverá permitir a visualizar a simulação a partir de um arquivo de Log.	Requisito previsto inicialmente e realizado
O CTA deverá permitir fazer a medição de tempo de execução de uma estratégia.	Requisito previsto inicialmente e realizado

Para o desenvolvimento do simulador CTA, iniciou-se pelo diagrama de classes em UML. Este foi dividido em quatro pacotes *Simulator*, *Controller*, *AsbstractFactory* e *Inteface*. Também o projeto foi divido em três threads implementadas através da biblioteca *Posix,* na qual as classes do pacote *Simulator*, *Controller* e *Interface* estão em thread separadas, enquanto as classes do pacote *AbstractFactory*, têm a funcionalidade apenas da criação.

O pacote *Simulator* contém a representação dos elementos do mundo real, como veículos, ruas, quadras, sinaleiros sob a forma de classes. Estas classes são responsáveis pela simulação do tráfego de uma região. A figura 3 apresenta o diagrama de classe deste pacote.

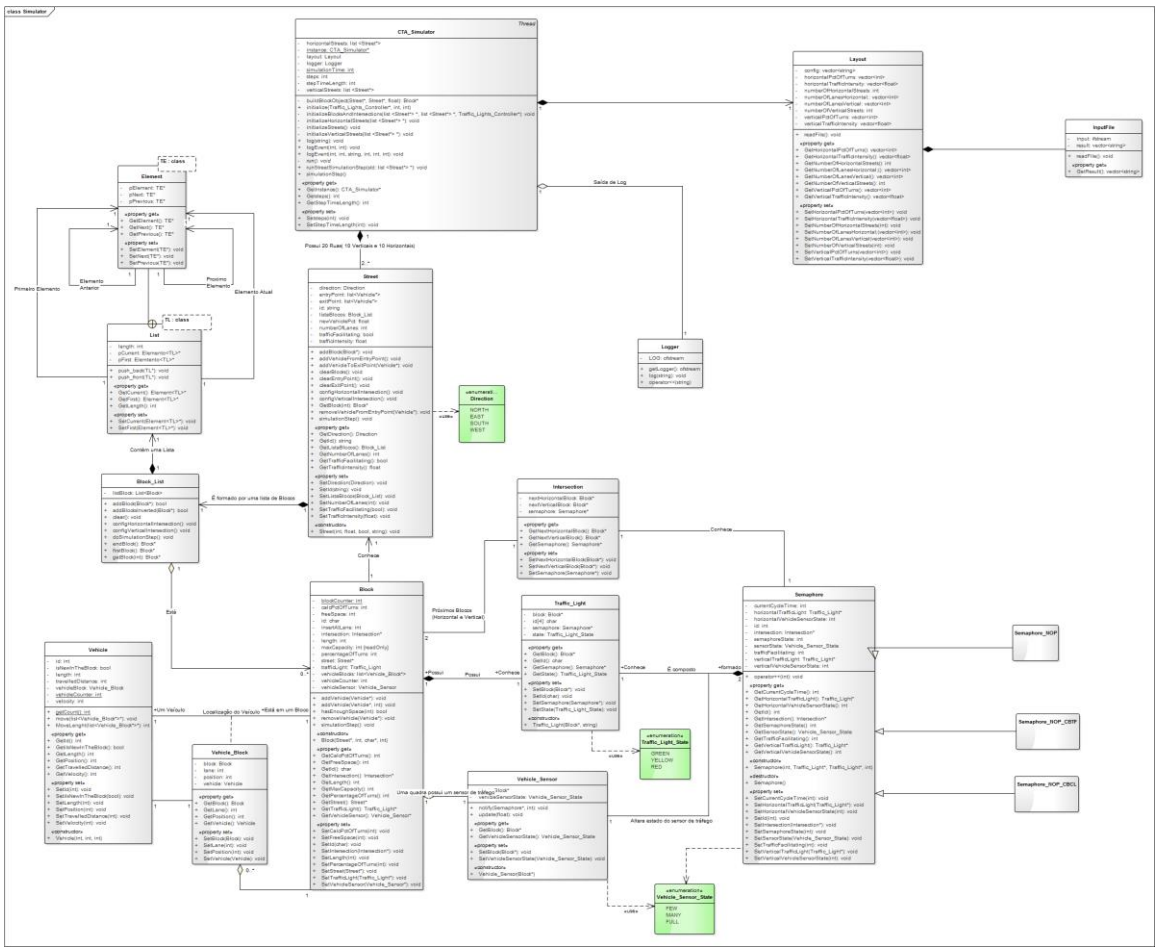

Figura 3. Diagrama de classes do pacote *Simulator*.

O pacote *Controller* contém as classes que são responsáveis pelas estratégias de controle dos semáforos. Este módulo conhece todos os semáforos instanciados e é configurado para utilizar diferentes estratégias de controle. Na análise de projeto foram levantadas três alternativas de controle, onde cada uma delas é implementada em dois paradigmas de programação diferentes, paradigma imperativo e paradigma orientado a notificações [1], com um total de seis estratégias diferentes. A figura 4 apresenta o diagrama de classes deste pacote.

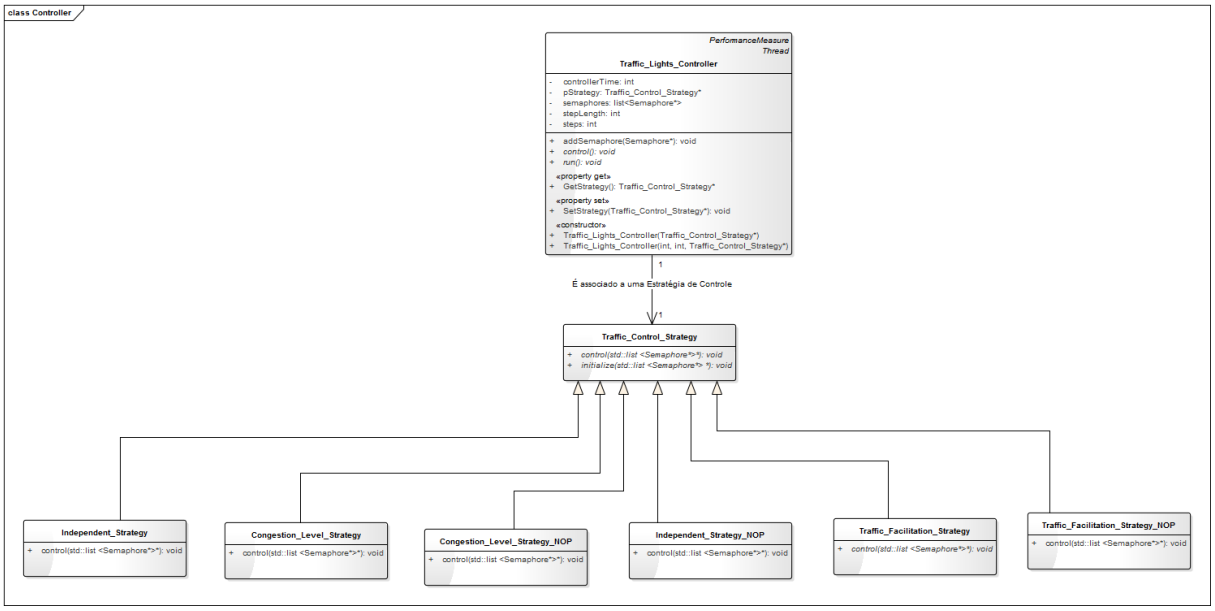

Figura 4. Diagrama de classes do pacote *Controller*.

O pacote *AbstractFacory* contém a interface de criação dos objetos necessários para cada cenário de simulação. Foram levantadas seis famílias de produtos concretos, na qual se pode variar o tipo de interface, semáforo e estratégia. A figura 5 apresenta o diagrama de classes do pacote.

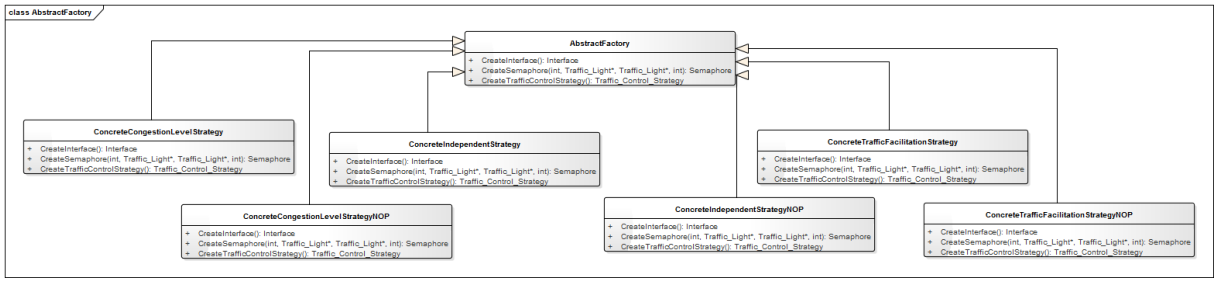

Figura 5. Diagrama de classes do pacote *AbstractFactory*.

O pacote *Interface* contém as classes responsáveis por implementar a interface gráfica da aplicação. Optou-se por usar Windows Forms como interface gráfica, porém seria possível implementar com outras bibliotecas, derivando da classe Interface e criando uma nova família de produtos no pacote *AbstractFactory*. A figura 6 apresenta o diagrama de classes do pacote *Interface*.

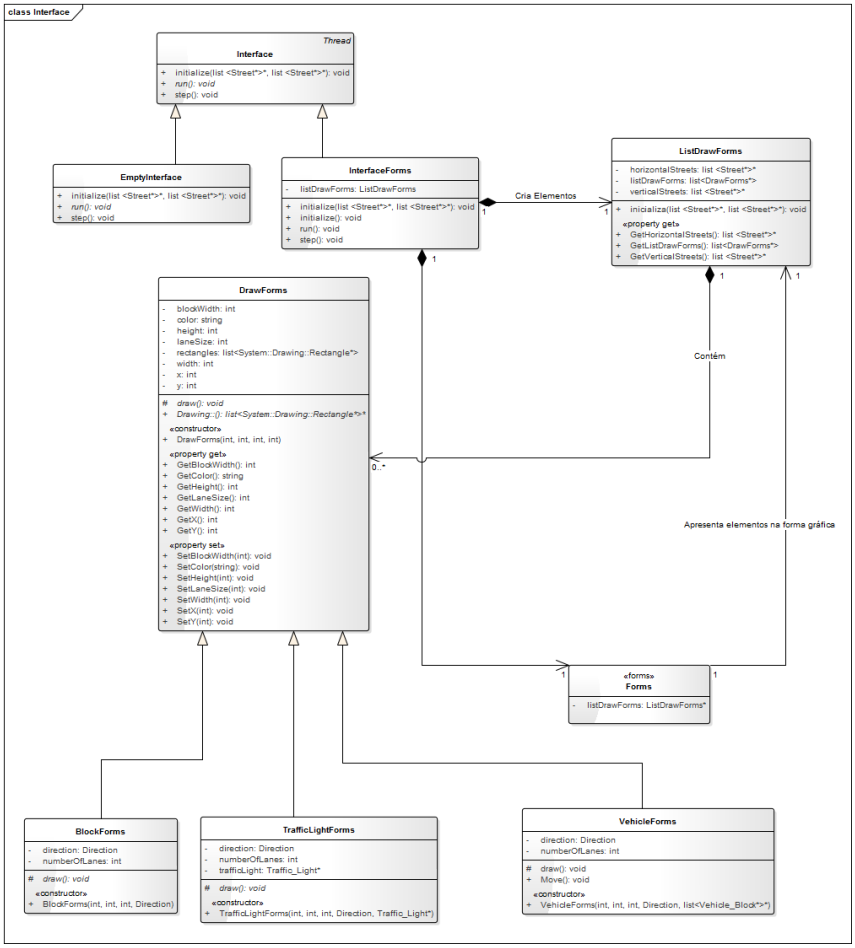

Figura 6. Diagrama de classes do pacote *Interface*.

Para lógica do controle independente foi utilizado o diagrama de estados como mostra na figura 7, conforme retratado no documento CONOPS [3]. Neste controle segue as regras apresentadas na seção II, de forma que cada semáforo controla seu tempo de forma independente e garantindo que não ocorram dois sinais verdes ao mesmo tempo.

Figura 7. Diagrama de estados para controle independente.

Na implementação da lógica de controle baseada em congestionamento, foi utilizada como referência a lógica apresentada na seção II, no qual o tempo de abertura de um semáforo passa a depender do estado do sensor de tráfego e do tempo do semáforo. O diagrama da figura 8 tem como referência o diagrama de estados da figura 7, assim foram adicionados novos estados ao diagrama para representar as mudanças de tempo de abertura do semáforo, de acordo com os estados dos sensores.

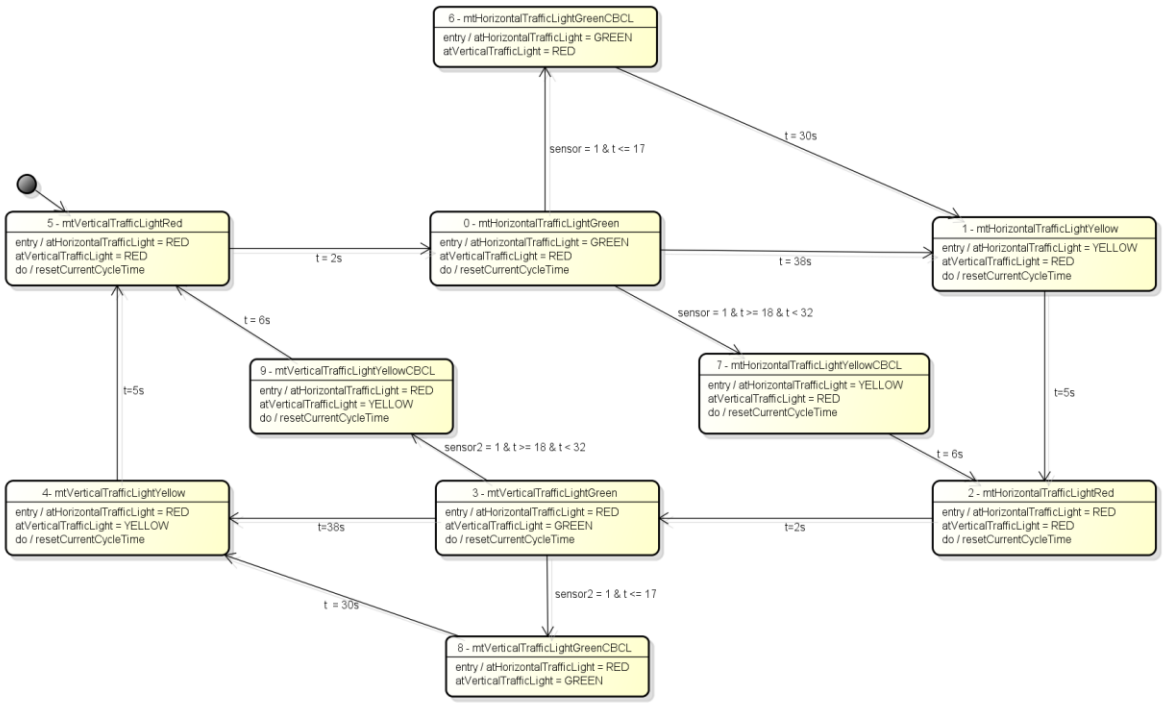

Figura 8. Diagrama de estados para controle baseado em congestionamento.

Para implementação da estratégia baseada em tráfego facilitado foi desenvolvido um novo diagrama de estados baseado no diagrama da figura 7 e na descrição da estratégia na Seção II. A figura 9 apresenta o novo diagrama de estados desenvolvido para a estratégia baseada em congestionamento. Este novo diagrama diferencia-se do diagrama da figura 7, pelo fato que a comparação é feita através do operador maior igual ao invés do operador somente de igualdade e foram adicionadas novas máquinas de estados para fazer a adição ou a subtração de tempo em relação aos semáforos vizinhos e seus respectivos sensores de tráfego.

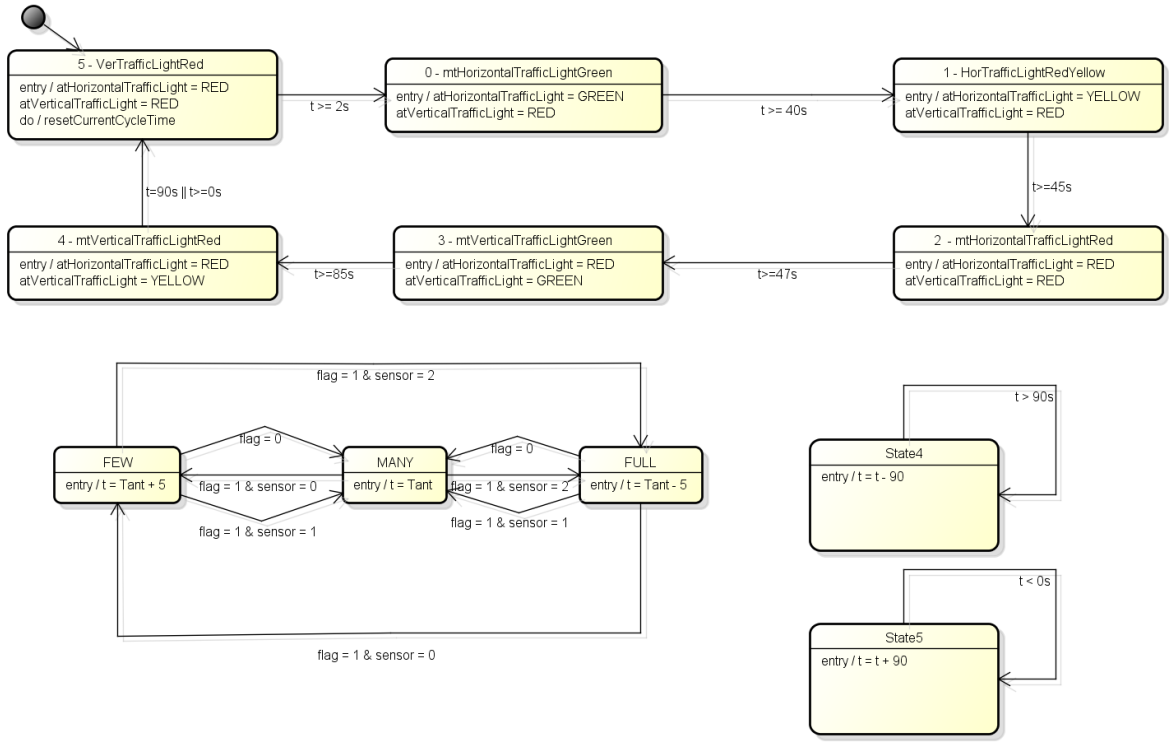

Figura 9. Diagrama de estados para controle baseado em tráfego facilitado.

A figura 10 apresenta a interface do software de configuração CTAGenerator após seu desenvolvimento.

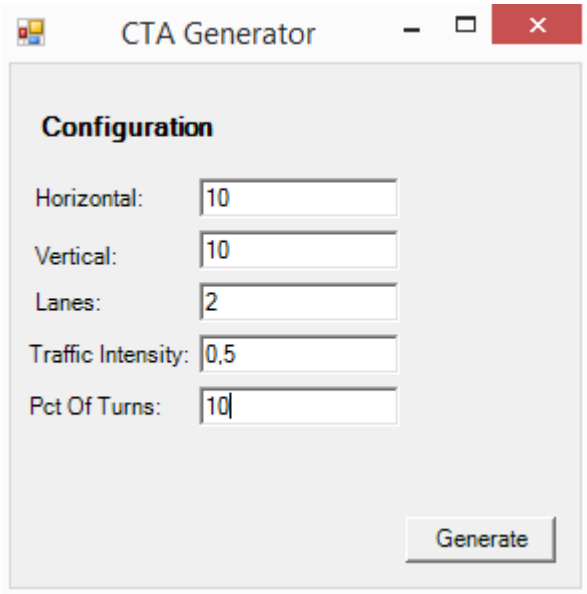

Figura 10. Interface Gráfica do CTAGenerator.

A figura 11 apresenta a interface do sistema CTA Simulator após seu desenvolvimento. No qual os elementos em cinza representam as ruas, os elementos verde e vermelho os semáforos em seus respectivos estados e os elementos em azul representam os veículos.

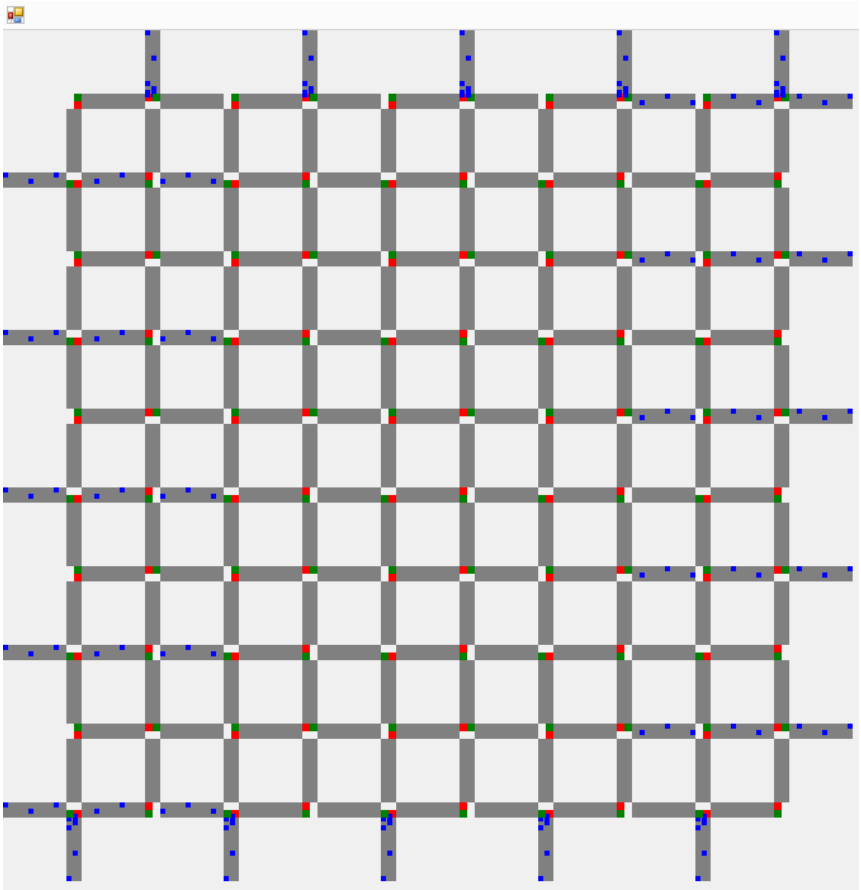

Figura 11. Interface Gráfica do CTA Simulator.

IV. TABELA DE CONCEITOS UTILIZADOS E NÃO UTILIZADOS

Nesta seção são apresentados os conceitos utilizados e não utilizados no desenvolvimento deste trabalho, através da Tabela 2. A Tabela 3 apresenta as justificativas com relação a utilização de cada conceito.

Tabela 2. Lista de Conceitos Utilizados e Não Utilizados no Trabalho.

N.	Conceitos	Uso	Onde
1	**Elementares:**		
	- Classes, objetos,	Sim	Todos .h e .cpp
	- Atributos (privados), variáveis e constantes	Sim	Todos .h e .cpp
	- Métodos (com e sem retorno).	Sim	Todos .h e .cpp
	- Métodos (com retorno *const* e parâmetro *const*).	Sim	Todos .h e .cpp
	- Métodos recursivos.	Não	-
	- Construtores (sem/com parâmetros) e destrutores	Sim	Todos .h e .cpp
	- Classe Principal.	Sim	CTA.h
	- Divisão em .h e .cpp.	Sim	No projeto.
2	**Relações de:**		
	- Associação	Sim	Intersection.h Vehicle.h Vehicle_Block.h Vehicle_Sensor.h Interface.h
	- Agregação via associação	Sim	Block_List.h Traffic_Light.h Block.h Street.h
	- Agregação propriamente dita.	Sim	CTA_Simulator.h Semaphore.h
	- Herança elementar.	Sim	Traffic_Control_Strategy.h Semaphore.h
	- Herança em diversos níveis.	Não	-
	- Herança múltipla.	Sim	Traffic_Lights_Controller.h
3	**Ponteiros e generalizações:**		
	- Operador *this*	Sim	Vehicle.h
	- Alocação de memória (*new* & *delete*)	Sim	CTA_Simulator.h Street.h Fábricas Concretas
	- Listas Encadeadas (bem Formadas)	Sim	List.h
	- Gabaritos/*Templates* criada/adaptados pelos autores	Sim	List.h
	Persistência de Objetos		
	- Texto via Arquivos de Fluxo	Sim	Logger.h
	- Binário	Não	-
4	**Sobrecarga de:**		
	- Construtoras e Métodos.	Sim	Todos.h
	- Operadores (2 tipos de operadores pelo menos)	Sim	Vehicle.h Independent_Strategy.h
	Virtualidade:		

	- Métodos Virtuais.	Sim	Semaphore.h Thread.h Traffic_Control_Strategy.h Interface.h AbstractFactory.h DrawForms.h
	- Polimorfismo	Sim	Traffic_Control_Strategy.h Interface.h Semaphore.h DrawForms.h
	- Métodos Virtuais Puros / Classes Abstratas	Sim	AbstractFactory.h DrawForms.h Interface.h
5	**Engenharia de Software**		
	- Levantamento de Requisitos Textualmente - E/ou via Diagrama de Casos de Uso em *UML*	Sim	
	- Diagrama de Classes em *UML*	Sim	
	- Diagrama de Atividades em *UML* - E/ou Fluxograma.	Não	
	- Outros diagramas em *UML*, Diag. de Estados, Diag. de Seqüência, Diag. de Pacotes etc. - E/ou outros diagramas estabelecidos, como Diag. de Fluxo de Dados (DFD) ou Diag. em SysML (Diag. de Requisitos,. de Blocos etc).	Sim	Diagramas de Estados.
6	**Biblioteca Gráfica**		
	- Funcionalidades Elementares.	Sim	InterfaceForms.h DrawForms.h
	- Funcionalidades Avançadas como: • tratamento de colisões	Sim	Vehicle.h InterfaceForms.h
	Obs.: especificar quais funcionalidades.		
	Interdisciplinaridades por meio da utilização de Conceitos de Matemática, Física etc		
	- Ensino Médio*	Sim	Vehicle.h
	- Ensino Superior*	Não	
	**Obs.: especificar quais Conceitos*		
7	**Organizadores:**		
	Espaço de Nomes (*Namespace*) criada pelos autores.	Não	
	Classes aninhadas.	Sim	List.h
	Estáticos e String:		
	Atributos estáticos e chamadas estáticas de métodos.	Sim	
	A classe Pré-definida String.Conceito.	Sim	
8	**Standard Template Library *(STL)***		
	Vector da STL (p/ objetos ou ponteiros de objetos de classes definidos pelos autores).	Sim	Street.h
	List da STL (p/ objetos ou ponteiros de objetos de classes definidos pelos autores).	Sim	Block.h Interface.h CTA_Simulator.h DrawForms.h
	*Pilhas, Filas, Bifilas, Filas de Prioridade, Conjuntos, Multi-Conjuntos, Mapas **ou** Multi-Mapas**.*	Não	
	*Obs.: Listar apenas os utilizados		
9	**Programação orientada a eventos e visual:**		
	*Objetos gráficos como formulários, botões etc**	Sim	CTAGenerator
	*Obs.: Listar apenas os utilizados		

10	**Programação concorrente:**		
	*Linhas de Execução (*Threads*) utilizando* Posix, C-Run-Time ***ou*** Win32API.	Sim	Thread.h CTA_Simulator.h Traffic_Lights_Controller.h Interface.h
	Mutex, Semáforos, **ou** Troca de mensagens.	Sim	Thread.h
	Threads *no âmbito da Orientação a Objetos*	Sim	Thread.h

Tabela 3. Lista de Justificativas para Conceitos Utiliza e Não Utilizados no Trabalho.

No.	Situação
1	Os conceitos elementares da programação orientada a objetos foram utilizados, pois são conceitos mais básicos para uma boa implementação. O único conceito não utilizado foi o de recursividade, pois não se demonstrou útil na implementação atual.
2	Foram utilizados todos os conceitos de relações entre classes.
3	Foram utilizados os conceitos de ponteiros e generalizações. Porém com relação a persistência de arquivos foi utilizado apenas texto via arquivos de fluxo, pois a geração de arquivo de log também foi usada como teste, assim a visualização é mais fácil do que em arquivo binário. Para a geração do arquivo de configuração utilizou-se de texto de arquivo para facilitar os testes.
4	Foram utilizados todos os conceitos de sobrecarga e virtualidade.
5	Dos conceitos não foram utilizados apenas o Diagrama de Atividades, porém foram feitos Diagrama de Estados para representar a lógica dos controles de semáforos.
6	Foi utilizado Windows Forms como interface gráfica. Também foi necessário tratar colisões entre veículos.
7	Foram utilizadas classes aninhadas para elaboração das listas encadeadas. Porém não foi aplicado o uso de namespace.
8	Foram utilizados os conceitos list e vector da STL, porém não foi necessário implementações com filas, pilhas, hashs, etc.
9	Foi utilizada programação visual e orientada a eventos para criar a aplicação de configuração da região de simulação.
10	Foram criadas três threads, uma para o simulador, estratégias de controle e uma para interface gráfica.

V. DISCUSSÃO E CONCLUSÕES

O desenvolvimento do projeto CTA Simulator permitiu aplicar diversos conceitos da orientação a objetos vistos na disciplina de programação avançada. Alguns conceitos já eram conhecidos antes da disciplina de programação avançada, devido à outra disciplina de programação cursada durante a graduação (Fundamentos de Programação 1 e 2). Porém muitos destes conceitos ainda não eram bem aplicados. Com a disciplina de programação avançada foi possível explorar melhor os conceitos de coesão e desacoplamento, e aplicar padrões de projetos.

O desenvolvimento do simulador de trânsito CTA se deu através do modelo cascata, no qual, primeiro foram levantados os requisitos, em seguida foram utilizados diagramas da UML, mais especificamente diagramas de classes e diagrama de estados. Na questão de diagrama de classes, a disciplina também permitiu o aprendizado da ferramenta CASE *Enterprise Architect*. Com os diagramas de classes foi possível melhorar a análise do sistema

como um todo e apontar melhorias com relação à coesão e desacoplamento, além de identificar potenciais usos de padrões de projetos e utilizar de maneira correta os conceitos da orientação a objetos. Após os diagramas validados, foi dado inicio ao desenvolvimento de código. Mesmo o projeto sendo iniciado em disciplinas anteriores, todos os códigos foram reescritos. A última fase do projeto foi a realização de testes através da interface gráfica e análise de arquivo de log.

O foco deste trabalho foi a elaboração de uma aplicação para a comparação entre o paradigma imperativo, através da linguagem de programação C++, com o paradigma orientado a notificações, através de uma linguagem própria denominada LingPON. Os resultados dessas comparações foram realizados e demostrado através de um artigo para a disciplina específica [1]. Após a reengenharia desta aplicação, foram realizados alguns experimentos e comparados com os resultados anteriores. Esses resultados foram muito próximos aos resultados da aplicação anterior. Porém, foi possível avançar com alguns requisitos que não haviam sidos implementados anteriormente, além de um melhor desacoplamento da estrutura da aplicação, permitindo a implementação de novos requisitos. Como trabalho futuro deste projeto, é possível avaliar novas implementações de estratégias de controle para comparação entre paradigmas. O paradigma orientado a notificações é um paradigma que ainda está em evolução quanto ao seu estado da técnica, portanto o CTA Simulator é uma alternativa para essas comparações.

REFERÊNCIAS

[1] L. F. PORDEUS, A. F. RONSZCKA, C. A. FERREIRA, R. R. LINHARES, J. A. FABRO, D. P. B. RENAUX, P. C. STADZISZ, J. M. SIMÃO, “Comparação entre Paradigma Orientado a Notificações e Paradigma Imperativo sobre um Simulador de Tráfego.”

[2] GAMMA, ERICH. Padrões de projeto: soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000 xii,364 p. ISBN 8573076100

[3] D. P. B. RENAUX, R. R. LINHARES, J. M. SIMÃO, P. C. STADZISZ , “CTA_CONOPS”, Available in: < http://www.dainf.ct.utfpr.edu.br/~douglas/CTA_CONOPS.pdf>, 2014,Accessed in July15th, 2015