TEN | Traffic Engine

Projeto de Software

**Versão 1.00**

17/06/2009

# Responsáveis

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Antônio Cláudio Goméz de Sousa

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Fernando Seabra Chirigati

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Rafael Shinji Aoki Kikuchi

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Talita Lopes Gomes

# Relatório de Mudanças

TEN – Traffic Engine

**Versão 1.00 – 17/06/2009**

Criação do Documento

# Índice

[1. Introdução 1](#_Toc232964729)

[1.1 Finalidade 1](#_Toc232964730)

[1.2 Escopo 1](#_Toc232964731)

[1.3 Definições, Acronismos e Abreviaturas 1](#_Toc232964732)

[2. Referências 2](#_Toc232964733)

[3. Decomposição 3](#_Toc232964734)

[3.1 Decomposição em Módulos 3](#_Toc232964735)

[3.1.1 Simulador 3](#_Toc232964736)

[3.1.2 Interface 3](#_Toc232964737)

[3.1.3 Atualizador 3](#_Toc232964738)

[3.1.4 Dados do Mapa 3](#_Toc232964739)

[3.1.5 Dados dos Veículos 3](#_Toc232964740)

[3.2 Decomposição em Processos Concorrentes 4](#_Toc232964741)

[3.2.1 Interface 4](#_Toc232964742)

[3.2.2 Simulação 4](#_Toc232964743)

[3.2.3 Controle de Redesenho da Interface 4](#_Toc232964744)

[3.3 Decomposição de Dados 4](#_Toc232964745)

[4. Descrição das Dependências 5](#_Toc232964746)

[4.1 Dependência entre módulos 5](#_Toc232964747)

[4.2 Dependência entre processos 5](#_Toc232964748)

[4.3 Dependência entre dados 6](#_Toc232964749)

[4.4 Diagramas de Seqüência 6](#_Toc232964750)

[4.5 Diagrama de Pacotes 8](#_Toc232964751)

[5. Descrição das Interfaces 10](#_Toc232964752)

[5.1 Interfaces dos Módulos 10](#_Toc232964753)

[5.1.1 Interface com o usuário 10](#_Toc232964754)

[5.1.2 Interfaces Internas 10](#_Toc232964755)

[5.1.3 Interfaces Externas 10](#_Toc232964756)

[5.2 Interfaces entre Processos 10](#_Toc232964757)

[5.2.1 TenApp 10](#_Toc232964758)

[6. Projeto Detalhado 11](#_Toc232964759)

[6.1 Projeto Detalhado dos Módulos 11](#_Toc232964760)

[6.1.1 Simulador 11](#_Toc232964761)

[6.1.2 Interface 11](#_Toc232964762)

[6.1.3 Atualizador 11](#_Toc232964763)

[6.1.4 Dados do Mapa 11](#_Toc232964764)

[6.1.5 Dados dos Veículos 12](#_Toc232964765)

[6.2 Projeto Detalhado das Entidades de Dados 12](#_Toc232964766)

# Introdução

## Finalidade

Este documento visa especificar os módulos, os processos e as estruturas de dados pertencentes ao TEN – Traffic Engine –, cujo ciclo de vida segue o Modelo de Processo em Cascata. Dessa maneira, forma-se uma base mais detalhada para o desenvolvimento do software, ajudando a equipe na atividade de codificação.

Além da equipe de desenvolvimento do sistema, a audiência deste documento é composta pelo gerente de qualidade do projeto, o professor Antônio Cláudio Goméz de Sousa.

## Escopo

O TEN – Traffic Engine – tem como principal objetivo a simulação de tráfego urbano em uma região confeccionada pelo próprio usuário através da interface. Logo, o software deve prover todas as ferramentas necessárias ao desenho dessa região, assim como todas as funcionalidades relacionadas à simulação. Além disso, a geração de um relatório referente a essa simulação pertence ao escopo do sistema.

## Definições, Acrônimos e Abreviaturas

Não se aplica.

# Referências

* *Plano de Gerenciamento de Projeto de Software – PGPS*

Versão: 1.00

Data: 01/04/2009

Responsável: Equipe de desenvolvimento do projeto

* *Especificação de Requisitos de Software – ERS*

Versão: 1.10

Data: 17/06/2009

Responsável: Equipe de desenvolvimento do projeto

* *Manual do Usuário – 2ª versão*

Versão: 1.00

Data: 17/06/2009

Responsável: Equipe de desenvolvimento do projeto

# Decomposição

## Decomposição em Módulos

O aplicativo será dividido em cinco módulos: Simulador, Interface, Atualizador, Dados do Mapa e Dados dos Veículos. Abaixo se encontra um diagrama que representa essa divisão.



A seguir, cada um dos módulos será descrito.

### Simulador

Este módulo é responsável pelos algoritmos de simulação de tráfego. As estruturas necessárias à realização da simulação estão presentes neste módulo, de forma que a alimentação de dados é feita através dos módulos **Dados do Mapa** e **Dados dos Veículos**.

### Interface

Este módulo é responsável por manter uma interface gráfica com o usuário, onde são exibidos os controles gráficos – janelas, botões, menu e outros –, a partir dos quais são feitas todas as interações com o aplicativo.

### Atualizador

O Atualizador é responsável por enviar pedidos de atualização à interface automaticamente durante a execução de uma simulação. Os pedidos são feitos periodicamente a fim de manter uma sensação de cadência ao usuário.

### Dados do Mapa

O módulo Dados do Mapa, como o nome sugere, mantém as estruturas de dados que contêm as informações que compõem o mapa gerado pelo usuário. Como estes dados são acessados por processos diferentes de forma potencialmente simultânea, é necessário que os métodos de acesso apresentem um controle de concorrência.

### Dados dos Veículos

O módulo Dados dos Veículos, de maneira semelhante ao módulo Dados do Mapa, também agrega estruturas de dados as quais contêm as informações necessárias à identificação de cada veículo da simulação. Também é necessário manter um controle de concorrência para acessar estes dados.

## Decomposição em Processos Concorrentes

O sistema pode ser dividido em três processos concorrentes: o primeiro é responsável pela interface, o segundo pela simulação e o último deles encarrega-se do controle de redesenho da interface durante a execução de uma simulação. Cada um desses processos é melhor explicado abaixo.

### Interface

Este processo é responsável pela manutenção da interface gráfica e por cada evento disparado de suas possíveis interações com o usuário. Como é necessário acessar dados que também são usados pelo processo da Simulação, é preciso fazer um controle de concorrência.

### Simulação

Este processo é responsável pela aplicação de algoritmos de simulação sobre os dados do mapa e os dados dos veículos, que devem ser acessados levando em consideração a concorrência com o processo da Interface.

### Controle de Redesenho da Interface

Este processo é responsável pelo controle de redesenho da interface gráfica durante uma simulação: nem toda iteração da simulação implica em um redesenho. Este controle é realizado com intuito de reduzir o custo computacional do sistema evitando uma sobrecarga de redesenhos.

## Decomposição de Dados

A decomposição de dados apresenta-se em conjunto com a decomposição em módulos, especificada anteriormente na seção 3.1. Além disso, uma descrição detalhada das entidades encontra-se na seção 3.2.6 da Especificação de Requisitos de Software (Dicionário de Dados).

# Descrição das Dependências

## Dependência entre módulos

As dependências entre os módulos podem ser vistas na figura abaixo. Na figura, se um determinado módulo X aponta para um módulo Y, indica que X depende de Y.



As dependências estão melhor especificadas a seguir:

* O módulo Interface depende das informações dos módulos Dados do Mapa e Dados dos Veículos para que a simulação do tráfego urbano possa ser reproduzida para o usuário – os dados são acessados através de métodos públicos;
* O módulo Atualizador depende do módulo Simulador para que seja verificada a necessidade de envio de redesenho para a interface gráfica, de modo que ela fique atualizada para o usuário – esta verificação é feita acessando uma propriedade pública para leitura;
* Para que os algoritmos de simulação sejam aplicados, é necessário que o módulo Simulador receba continuamente as informações dos módulos Dados do Mapa e Dados dos Veículos, acessadas por métodos públicos.

## Dependência entre processos

As dependências entre os processos podem ser vistas na figura abaixo. Além do tipo de representação visto acima, nesta figura há também uma seta bidirecional pontilhada, que indica a existência de uma região de exclusão mútua entre os processos.



As dependências estão melhor explicadas a seguir:

* Simulação e Controle de Redesenho da Interface: o controlador de redesenho – representado pelo módulo Atualizador – só deve enviar pedidos de atualização da interface enquanto a Simulação estiver ativa.
* Simulação e Interface: tanto a Simulação quanto a Interface acessam dados em comum passíveis de alteração e, portanto, deve haver um controle de concorrência para o acesso a seções críticas. Para isso, há um método através do qual o acesso a estes dados é feito de forma mutuamente exclusiva – para esta implementação, a declaração *lock* do C# é usada.

## Dependência entre dados

A dependência entre os dados é mostrada anteriormente na dependência entre os módulos, que se encontra no item 4.13.1 deste documento. O Diagrama de Classes presente no item 3.2.5 da Especificação de Requisitos de Software também apresenta informações relevantes à dependência entre os dados.

## Diagramas de Seqüência

O diagrama de seqüência mostra claramente como as classes colaboram de maneira a executar um determinado caso de uso. Na Especificação de Requisitos de Software, diagramas de seqüência referentes a três estudos de casos, os quais também estão especificados no mesmo documento, foram apresentados, porém sem considerar as classes e os métodos do sistema.

Nesta seção, os mesmos diagramas são mostrados, mas agora contendo maiores detalhes em relação às classes e aos métodos.

#### Caso de Uso 1 – Criar Rua

#### 

#### Caso de Uso 7 – Semaforizar um Ponto



#### Caso de Uso 11 – Iniciar Simulação



## Diagrama de Pacotes

O diagrama de pacotes é importante para descrever como os sub-sistemas do software são divididos em agrupamentos lógicos de classes e objetos coesos, mostrando as dependências existentes.

O diagrama de pacotes do TEN encontra-se a seguir.



# Descrição das Interfaces

## Interfaces dos Módulos

### Interface com o usuário

A interface com o usuário é dada pela interface gráfica propriamente dita, fornecida pelo módulo Interface.

### Interfaces Internas

As interfaces internas entre os módulos são realizadas através de chamadas de métodos públicos.

### Interfaces Externas

Não há interfaces externas.

## Interfaces entre Processos

A comunicação entre os processos é feita através de propriedades estáticas de classes em comum.

### TenApp

A classe estática TenApp possui propriedades estáticas que fazem referência às classes principais de cada processo concorrente. Desta forma, cada processo pode acessar todas as propriedades e métodos públicos inerentes a objetos associados a outros processos.

# Projeto Detalhado

## Projeto Detalhado dos Módulos

Nas seções abaixo serão descritos os métodos referentes a cada um dos módulos.

### Simulador

* void Start();

Método chamado para iniciar ou retomar um processo de simulação, aplicando os algoritmos do software. O Atualizador é ativado para realizar o redesenho periódico da interface. Caso já existam dados de simulação sendo usados, eles são retomados para dar seguimento à simulação.

* void Stop();

Método usado para interromper a simulação e limpar os dados associados a ela – veículos, contadores internos, dados de relatórios, entre outros.

* void Pause();

Método chamado para pausar uma simulação em andamento, mantendo os dados da simulação para que seja possível retomá-la.

* void Restart();

Método usado para reiniciar a simulação, interrompendo a que está em andamento e limpando seus dados.

### Interface

* void SetState(AppState newState);

Define um novo estado para a interface, habilitando e desabilitando os botões necessários de acordo com o novo estado. Caso o usuário clique no botão de iniciar a simulação, por exemplo, os botões de adicionar ruas e semáforos são desativados.

* void Draw();

Método chamado para redesenhar a região da interface referente à simulação.

### Atualizador

* void Run();

Método que ativa o processo de controle de redesenho da interface de modo que pedidos de atualização sejam enviados periodicamente a ela.

* void Stop();

Desativa as chamadas de atualização enviadas pelo processo do Atualizador, a fim de evitar a sobrecarga de processamento enquanto não há necessidade de redesenhos.

### Dados do Mapa

* List<MapEdge> GetEdges();

Retorna o conjunto de arestas presentes no mapa através de um método seguro para threads concorrentes – é possível chamar o método sem se preocupar com problemas de concorrência.

* List<MapNode> GetNodes();

Retorna o conjunto de nós contidos no mapa através de um método seguro para threads concorrentes.

* List<FlowNode> GetFlowNodes();

Retorna o conjunto de objetos FlowNode que o mapa possui através de um método seguro para threads concorrentes.

* List<Semaphore> GetSempahores();

Retorna o conjunto de semáforos contidos no mapa através de um método seguro para threads concorrentes.

### Dados dos Veículos

* List<Vehicle> GetVehicles();

Retorna o conjunto de veículos da simulação em andamento por um método seguro para threads concorrentes.

## Projeto Detalhado das Entidades de Dados

Não se aplica.