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

[Responsáveis i](#_Toc232778528)

[Relatório de Mudanças ii](#_Toc232778529)

[Índice iii](#_Toc232778530)

[1. Introdução 1](#_Toc232778531)

[1.1 Finalidade 1](#_Toc232778532)

[1.2 Escopo 1](#_Toc232778533)

[1.3 Definições, Acronismos e Abreviaturas 1](#_Toc232778534)

[2. Referências 2](#_Toc232778535)

[3. Decomposição 3](#_Toc232778536)

[3.1 Decomposição em Módulos 3](#_Toc232778537)

[3.1.1 Simulador 3](#_Toc232778538)

[3.1.2 Interface 3](#_Toc232778539)

[3.1.3 Atualizador 3](#_Toc232778540)

[3.1.4 Dados do Mapa 3](#_Toc232778541)

[3.1.5 Dados dos Veículos 3](#_Toc232778542)

[3.2 Decomposição em Processos Concorrentes 3](#_Toc232778543)

[3.2.1 Interface 3](#_Toc232778544)

[3.2.2 Simulação 4](#_Toc232778545)

[3.2.3 Controle de redesenho da interface 4](#_Toc232778546)

[3.3 Decomposição de Dados 4](#_Toc232778547)

[4. Descrição das Dependências 5](#_Toc232778548)

[4.1 Dependência entre módulos 5](#_Toc232778549)

[4.2 Dependência entre processos 5](#_Toc232778550)

[4.3 Dependência entre dados 5](#_Toc232778551)

[4.4 Diagramas de Seqüência 5](#_Toc232778552)

[4.5 Diagrama de Pacotes 7](#_Toc232778553)

[5. Descrição das Interfaces 9](#_Toc232778554)

[5.1 Interfaces dos Módulos 9](#_Toc232778555)

[5.1.1 Interfaces do módulo 1 9](#_Toc232778556)

[5.2 Interfaces entre Processos 9](#_Toc232778557)

[5.2.1 Interfaces do processo 1 9](#_Toc232778558)

[6. Projeto Detalhado 10](#_Toc232778559)

[6.1 Projeto Detalhado dos Módulos 10](#_Toc232778560)

[6.1.1 Detalhamento do módulo 1 10](#_Toc232778561)

[6.2 Projeto Detalhado das Entidades de Dados 10](#_Toc232778562)

[6.2.1 Detalhamento da entidade de dados 1 10](#_Toc232778563)

# 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, Acronismos 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: 1.00

Data: 18/05/2009

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

Os dados do mapa, como o nome sugere, mantêm 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

Os dados dos veículos, de maneira semelhante aos dados do mapa, também agregam 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 mais bem 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 de 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 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 são as seguintes:

* 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 são:

* Simulador e Controlador de redesenhos da Interface: o controlador de redesenho – representado pelo módulo Atualizador – só deve enviar pedidos de atualização da interface enquanto o simulador estiver ativo.
* Simulador e Interface: tanto o Simulador 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, será implementado um método através do qual o acesso a estes dados é feito de forma mutuamente exclusiva – para esta implementação, usa-se a declaração *lock* do C#.

## Dependência entre dados

A dependência entre os dados é mostrada anteriormente na dependência entre os módulos, que se encontra no item 4.1 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 abaixo.



# 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 possuirá 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 e 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.