Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e Computação
Departamento de Engenharia de Computação e Automação Industrial

UNICAMP

Modelagem Estruturada e Sistemas Inteligentes: Uma Aplicação a Sistemas de Transporte Ferroviário

Magali Mercedes Rondón González

Orientador: Prof. Dr. Fernando Antonio Campos Gomide

Banca Examinadora:

Prof. Dr. Fernando Antonio Campos Gomide
UNICAMP

Prof. Dr. Walmir Matos Caminhas
UFMG

Prof. Dr. Tsen Chung Kang
USP

Prof. Dr. Rafael Santos Mendes
UNICAMP

Prof. Dr. Ivan Luiz Marques Ricarte
UNICAMP

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação (FEEC) da Universidade Estadual de Campinas (UNICAMP), como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica.

Dissertação de Mestrado
Campinas – SP – Brasil
Setembro de 2000

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

R668m

Rondón González, Magali Mercedes
Modelagem estruturada e sistemas inteligentes: uma aplicação a sistemas de transporte ferroviário / Magali Mercedes Rondón González.--Campinas, SP: [s.n.], 2000.

Orientador: Fernando Antonio Campos Gomide
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Sistemas difusos. 2. Simulação (Computadores). 3. Trens de ferro. 4. Transporte ferroviário. 5. Sistemas inteligentes de controle. I. Gomide, Fernando Antonio Campos. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

A minha família

a quem amo acima de tudo

AGRADECIMENTOS

Ao professor Gomide por sua dedicação, paciência e grande importância na minha formação acadêmica;

A Cinthia por ter me fornecido o apoio inicial necessário para empreender este mestrado;

Ao Miguel por seu carinho, apoio, e companhia ao longo do desenvolvimento deste trabalho;

A Lizet, Luis e José Antonio por terem sido a minha família quando comecei o mestrado e por me ajudarem no processo de adaptação ao Brasil;

Ao Pedro e Carlos por terem me fornecido apoio, amizade e proteção quando o necessitava;

A meu amigo Rodrigo pela amizade e pelos conselhos que ajudaram no desenvolvimento deste trabalho;

À FAPESP pelo suporte financeiro;

Ao professor A.J. Taylor pela valiosa informação fornecida que serviu de base para este trabalho;

A Marina Aguado pelo fornecimento de dados que ajudaram no teste do trabalho;

Enfim, a todos aqueles que me ajudaram nestes anos de mestrado,

Agradeço

RESUMO

Modelos de linhas ferroviárias são essenciais para análise e decisão em sistemas ferroviários. Estes modelos fornecem meios para análise de movimento e despacho, fornecendo meios para simulação, análise de operações, otimização, controle de circulação e expansão de capacidade, entre outros problemas.

Este trabalho apresenta um modelo estruturado de linha que estende o modelo clássico proposto por Petersen e Taylor em 1982. O modelo proposto acrescenta funcionalidades que não foram consideradas no modelo original e foi construído com uma visão que privilegia a orientação a objetos. Como conseqüência, um sistema computacional foi implementado em linguagem Java com o intuito de fornecer uma ferramenta de análise e suporte na tomada de decisão.

Mais especificamente, o modelo aqui desenvolvido acrescenta as seguintes funcionalidades ao modelo original: criação de modelos de trens, percursos e escalas, circulação de trens numa malha ferroviária e perturbação de trens.

A ferramenta computacional fornece mecanismos para caracterizar modelos de linha, simular a circulação de trens em uma malha ferroviária, visualizar dados e obter resultados de simulação que traduzem o desempenho do sistema ferroviário. Além disso, a ferramenta permite definir e avaliar funções de otimização de despacho através de uma linguagem concebida para este propósito. Estas funções podem ser definidas tanto através de algoritmos convencionais quanto em algoritmos da inteligência computacional, com ênfase naqueles baseados na teoria de conjuntos nebulosos.

Exemplos de aplicação que servem para ilustrar o potencial do modelo e a utilização da ferramenta são também incluídos. Finalmente, as conclusões e as atividades que ainda necessitam de desenvolvimentos futuros são resumidas.

ABSTRACT

Railway line models are essential for analysis and decision making of rail systems. They provide a vehicle to analyze train movements and train dispatch strategies. Line models are also a key for system simulation, operation analysis, systems optimization and control, and capacity expansion as well.

This work extends the classic Petersen and Taylor model developed in 1982. The new model introduces functionalities that were not part of original model of Petersen and Taylor. For instance, the model developed here includes the following additional characteristics: train, journey and stop models, train circulation over rail network and train disturbances. The model was developed using an object-oriented approach, and translated into a decision support software tool. Besides system simulation, operation analysis, capacity expansion, and train movement analysis, the tool also provides a language especially suitable to define circulation optimization using both, conventional and computational intelligence based procedures, with a focus on fuzzy set theory algorithms.

Application examples are also included to illustrate the usefulness of the model and the software tool. Finally, the conclusions are summarized and future work suggested.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Sistemas de transporte são uma classe de sistemas que vem se beneficiando do desenvolvimento teórico e metodológico da inteligência computacional, principalmente nos aspectos relacionados com modelagem, controle, otimização e logística [PED98].

No caso particular de sistemas de transporte ferroviários, a lógica nebulosa vem sendo aplicada em controle de circulação [MIN94]; em problemas de suporte ao planejamento, otimização, controle e logística de tráfego combinados, entre outros.

Os sistemas de logística de transporte ferroviário se mostram como uma plataforma excepcionalmente rica para aplicar, testar e explorar metodologias e algoritmos de sistemas inteligentes, dada a diversidade e a complexidade dos problemas que se apresentam (e.g., elaboração de *schedulings* táticos, *re-scheduling* em tempo real, análise de operações, controle de circulação, análise de capacidade, expansão de capacidade, alocação de recursos, etc.). Para explorar estas metodologias e algoritmos nos sistemas de transporte ferroviário, é necessário desenvolver sistemas e obter modelos que sejam estruturados em consonância com a engenharia de sistemas inteligentes. Os modelos básicos envolvidos nos sistemas de transporte ferroviário incluem modelos de linhas, de pátios intermediários e de terminais. Embora cada um destes modelos sejam individualmente complexos, o modelo de linha é o primordial por prover, por si só, bases para estudos envolvendo simulação, análise de operações, otimização e controle de circulação, expansão de capacidade, dentre outros.

O objetivo principal deste trabalho é o de desenvolver um modelo de linha que seja consonante com a engenharia de sistemas inteligentes. Além disso, objetiva também a partir do modelo, implementar um sistema computacional capaz de simular diversas situações que envolvam questões relativas ao controle e à otimização via métodos de inteligência computacional.

1.1. Descrição do Problema

As linhas ferroviárias constituem um importante meio de transporte tanto de passageiros quanto de carga. A indústria ferroviária possui um grande capital investido em equipamentos e também é uma grande fonte geradora de empregos. Sua administração envolve um sistema complexo de tomada de decisões e planejamento.

Como em outros meios de transporte o sistema de transporte ferroviário pode ser comparado a uma rede [ASS80a]. As conexões da rede referem-se às linhas onde efetuam os movimentos dos trens e ocorrem as principais decisões operacionais que afetam o tráfego entre os pátios. Estas decisões operacionais estão relacionadas com atividades de linha e caracterizam o modelo de uma linha. As linhas podem ser singelas ou duplas e são divididas em segmentos que representam um trecho entre dois pontos de transferência, desvios, pátios ou estações. Um desvio em um linha singela é um pequeno segmento de trecho duplo entre longos segmentos de um trecho singelo que podem ser utilizados para os cruzamentos ou ultrapassagens dos trens.

Os nós da rede se referem às estações ou pátios que organizam o tráfego de entrada e saída dos trens. As operações realizadas em um pátio são chamadas de atividades de pátio e caracterizam o modelo de pátios. Todo pátio possui uma função de produção específica de acordo com as atividades de produção escalonadas e sua capacidade de processar a entrada de tráfego.

Os modelos de linhas analisam os movimentos dos trens e as atividades de despacho. Eles investigam a capacidade da linha, identificam e analisam as diferentes políticas de prioridade em ultrapassagens e cruzamentos. A capacidade da linha pode ser avaliada tomando como base os atrasos sofridos por trens sob diferentes políticas operacionais. Exemplos destes modelos podem ser encontrados em [PET74], [ENG77], etc.

Os modelos de pátios analisam o comportamento em um determinado tipo de pátio (intermediário ou terminal). Eles são usados na avaliação dos efeitos da aplicação de certas políticas nos pátios e servem de ajuda na alocação de recursos e equipamentos. Estes modelos provêem uma estimativa do atraso médio, de custos operacionais e de pontos de congestionamento associados com as operações nos pátios. Um exemplo deste tipo de modelo pode ser encontrado em [SHI66].

Modelos que representam tanto atividades de pátios como atividades na linha podem ser observados em [THO71], [PET73] e [ASS80b].

Ambos modelos são relevantes na estrutura do sistema de transporte ferroviário. Apesar disso, este trabalho está dirigido ao modelo de linha, visto que este provê bases para estudos envolvendo simulação, análise de operações, otimização e controle de circulação, expansão de capacidade, dentre outros; podendo descrever os pátios intermediários e terminais de forma simples, porém representativos.

Os sistemas de transporte ferroviário vem sendo objeto de estudo há muitos anos dando lugar a criação de diversos modelos. Em um primeiro momento os resultados destes modelos estavam baseados na aplicação da matemática, ciência da computação clássica e algoritmos exatos. Não obstante, estes modelos apresentavam limitações com respeito à representação da incerteza implícita ao problema. Na atualidade, nos sistemas de transporte ferroviários vêm sendo aplicados métodos heurísticos que visam encontrar soluções rápidas fazendo uso da inteligência computacional e do conhecimento de especialistas. Com isso, muitas das limitações apresentadas inicialmente estão sendo contornadas. Exemplos da aplicação da inteligência computacional na resolução de problemas de transporte ferroviário podem ser observados em: [GOM99], [GON00], [MIN94], [RAM00], [VIE96] dentre outros.

Na atualidade, existe uma ausência de tecnologia nacional de suporte a problemas de decisão e de logística de interesse à nossa realidade, além da ausência, também em termos internacionais, de ferramentas que contemplem os avanços recentes da computação e da inteligência computacional. Incluem-se neste contexto o surgimento de técnicas modernas de desenvolvimento e simulação de sistemas, (e.g. Unified Modeling Language (UML) [HAR97] e SIMNET [TAH88]), linguagens (e.g. Java [FEL98]) entre outros aspectos.

Por esses motivos é relevante o desenvolvimento de um sistema computacional que leve em conta os avanços computacionais para construir modelos de linhas ferroviárias. Este sistema deverá implementar um modelo de linha que seja estruturado em consonância com as bases dos sistemas inteligentes e que descreva em forma simples, porém representativa, os pátios intermediários e terminais da linha ferroviária.

1.2. Motivação

A motivação decorre do recente renascimento, após longo período de injustificado abandono, das ferrovias nacionais e de sua relevância no contexto sócio-econômico do país, da ausência de tecnologia nacional de suporte à problemas de decisão e de logística de interesse à nossa realidade, além da ausência, também em termos internacionais, de ferramentas que contemplem os avanços recentes da computação e da inteligência computacional.

O transporte ferroviário possui um custo fixo elevado devido à exigência da construção de leitos elaborados e de material rodante, especialmente de locomotivas e vagões. Apesar de não apresentar grande flexibilidade por operar através de pontos fixos caracterizados por estações e pátios, é muito competitivo no transporte de cargas com origens e destinos fixos e para longas distâncias. Apresenta ainda baixo custo operacional e baixo consumo de combustível em relação ao transporte rodoviário.

Atualmente o Brasil possui aproximadamente 30.300 km de ferrovias para o tráfego, o que corresponde a uma densidade ferroviária de 3,1 metros por km^2; densidade muito pequena em relação aos Estados Unidos (150 metros por km^2). Além de curtas extensões, as ferrovias não estão uniformemente distribuídas, pois 47% delas concentram-se na região sudeste do país. Registra-se uma participação de 23% no total de movimentação de cargas.

Vários são os problemas que barram o desenvolvimento das ferrovias brasileiras: material rodante deficiente, concorrência das rodovias, alto custo de instalação e escassez de combustível. Em vista destes problemas, a maior parte da rede ferroviária no país é deficitária. Algumas atitudes governamentais estão sendo tomadas para solucionar problemas do transporte ferroviário, estando entre elas a: reorganização administrativa, privatizações, reaparelhamento das ferrovias e substituição de locomotivas por outras de maior rendimento. Apesar destas medidas, o transporte ferroviário ainda está longe de atingir um plano ideal condizente com a necessidade econômica [AMB99].

Na economia norte-americana o transporte ferroviário possui importância significativa, para o transporte de carga e de passageiros. Nos Estados Unidos ele é responsável por cerca de 33,6% do transporte fretado, transportando quase a metade dos produtos que circulam no país. No entanto, devido às competições de outras modalidades de transporte, a indústria ferroviária americana também exige uma maior eficiência operacional. Racionalizar decisões,

planejar, usar modelos de suporte à operação ferroviária com o objetivo de melhorar as funções de despacho de trens e o controle de tráfego na linha, tem sido os itens considerados para maior eficiência do transporte.

1.3. Definição de Objetivos

Os objetivos deste trabalho podem ser sumarizados da seguinte forma:

1) Pesquisa e estudo de modelos de linhas ferroviárias visando um detalhamento do estado da arte na área;
2) Pesquisa e estudo de técnicas de desenvolvimento de sistemas orientados a objetos usando a linguagem UML e de sua aplicação em desenvolvimento de modelos orientados a objetos de linhas, e implementação utilizando a linguagem Java;
3) Definição e formalização de um modelo estruturado de linha que contemple a existência de pátios intermediários, bem como a relação da linha com terminais;
4) Implementação computacional e testes do modelo através de um sistema que inclua interfaces apropriadas para a definição da topologia de linha e demais parâmetros que caracterizam o sistema físico, bem como a definição de estratégias de decisão baseadas em metodologias de inteligência computacional, particularmente aquelas baseadas em lógica nebulosa.

Em síntese, o objetivo principal é o de desenvolver um modelo orientado a objetos que seja descrito em um espaço de estados apropriado para resolver o problema de mover trens em uma linha, bem como uma descrição das relações algébricas e métodos que são necessários para garantir as considerações de factibilidade e segurança. A expectativa é que deste modelo resulte uma abordagem genérica e um sistema computacional para simulação e análise que proporcione a realização de estudos envolvendo questões relativas ao controle e à otimização operacional via métodos de inteligência computacional.

1.4. Organização do Texto

Este texto está organizado em seis capítulos, e da seguinte maneira: Após este capítulo introdutório, o capítulo 2 apresenta os conceitos básicos do modelo de linha e dos modelos de despacho de trens. Descreve-se a relevância do modelo de linha e sua relação com os

modelos de despacho. O capítulo 3 apresenta o modelo de linha proposto e o modelo de linha que serviu de base para sua formulação. O capítulo 4 apresenta a modelagem orientada a objetos do sistema computacional desenvolvido usando notação UML. O capítulo 5 apresenta a ferramenta computacional desenvolvida e mostra exemplos de seu uso. Finalmente, o capítulo 6 apresenta as conclusões do trabalho e sugere-se novas propostas para o futuro.

Capítulo 2

Modelos de Linha e de Despacho

Neste capítulo, será apresentada a revisão bibliográfica sobre modelos de linhas ferroviárias e modelos de despachos de trens. Estes modelos estão estritamente relacionados, pois a movimentação de trens na linha – e consequentemente os atrasos que estes possam sofrer – dependem diretamente da estratégia de despacho usada.

2.1. Modelos de Linha

Muitos modelos de linha têm sido propostos com a finalidade de estimar o atraso de cada trem causado por interferências em uma linha ferroviária. Estas interferências são produto de estratégias de despacho, distribuição do tráfego, topologia física da linha ferroviária e comportamento de terminais de carga, descarga e transbordo.

Um dos primeiros modelos foi desenvolvido por Frank [FRA66]. O modelo, aplica-se a linhas simples, com tráfego em ambos sentidos, trens viajando com uma mesma velocidade e desvios eqüidistantes. Um modelo mais elaborado foi proposto por Petersen [PET74]. Este modelo envolve trens viajando com diferentes velocidades e em ambas as direções. Também envolve desvios que permitem ultrapassagens e cruzamentos. Petersen [PET75] introduziu expressões algébricas que permitem determinar quando ocorrem atrasos devido a ultrapassagens e cruzamentos de trens. Um modelo de filas para determinar o atraso esperado devido a decisões de despacho foi apresentado por Greenberg *et al.* [GRE88]. O atraso é calculado para uma linha ferroviária simples, com trens viajando em baixa velocidade e um grande espaçamento entre os pátios. Kraft [KRA88] estendeu o enfoque de [PET75] levando em conta interações entre vários trens e comparando os resultados obtidos mudando a função objetivo utilizada na otimização do despacho.

Chen e Harker [CHE90] introduziram um modelo que contempla trens escalonados. A média e a variância dos tempos de viagem são estimados resolvendo um sistema de equações não lineares que também leva em consideração incertezas na hora de partida. Harker e Hong [HAR90] estenderam este modelo para linhas duplas.

Recentemente, Hallowell and Harker [HAL96] descreveram um modelo usado para predizer a performance dos trens considerando as chegadas e um horário predeterminado. Este modelo representa uma alternativa interessante aos modelos de simulação na estimação de atrasos de trens e pode ser aplicado a modelos de escalonamento de trens e/ou despacho dos mesmos.

Petersen e Talyor [PET82] propuseram um trabalho para modelar os movimentos dos trens sobre uma linha singela ou múltipla. Este modelo permite utilizar funções de otimização do despacho. Ozekici [OZE94] analisou o problema do despacho de trens com ênfases nos sistemas de transporte suburbano. O modelo é usado para avaliar a performance de diversas políticas de despacho tais como medida do atraso no serviço e do tempo médio de espera dos passageiros.

2.2. Modelos de Despacho de Trens

Os modelos de despacho de trens têm recebido maior atenção nos últimos tempos. O investimento em linhas ferroviárias trouxe consigo o desenvolvimento e implementação de sistemas avançados de controle de trens. Estes sistemas provêem informação em tempo real assim como decisões para auxílio às operações. Eles ajudam a diminuir o consumo de combustível, a incrementar a capacidade da linha e a confiança no serviço [COR98].

Petersen e Taylor [PET86] apresentaram um breve estudo sobre a evolução do despacho assistido por computador. Em um primeiro nível, o despacho assistido por computador libera o despachador de atividades rotineiras, embora as atividades críticas continuem sob a responsabilidade do despachador. A estas primeiras versões de despacho assistido por computador foram incorporadas funções adicionais como: definição do trajeto de um trem, liberação de sinais na frente do trem para evitar conflitos, determinação exata de ultrapassagens e cruzamentos, etc. Em um segundo nível, o despacho assistido por computador implementa lógicas de despacho para examinar conflitos e sugerir soluções. Estas lógicas podem ser regras padrão ou podem analisar o próximo, vários, ou todos os conflitos para proporcionar uma solução. Em um terceiro nível, o despacho é totalmente automatizado. Nele, o computador controla a decisão de despacho. A figura do despachador é removida da malha de controle, cabendo-lhe somente a tarefa de supervisão. Ele só participa do sistema quando ocorrem eventos e situações não previstas. Também existe o despacho integrado. Neste tipo de sistema, o despachador fornece constantemente

informações ao maquinista do trem com o objetivo de minimizar atrasos e consumo de combustível. A comunicação sincronizada entre o despachador e o maquinista faz com que o fator tempo seja uma restrição importante neste tipo de sistemas.

Graças à incorporação destes sistemas às linhas ferroviárias, o trabalho pesado do despachador foi aliviado, assim como também foram reduzidos os custos operacionais e a segurança aumentada. Como exemplo podemos considerar o caso da *Southern Railway Company* [SAU83]. Em 1983 instalou-se um sistema de despacho que utilizava programação inteira na tomada de decisões. Reportou-se uma diminuição de 15% nos atrasos dos trens o que eqüivalia a US$316.000,00 anuais. Outras vantagens que obtiveram foram:

- um trabalho mais claro, limpo e profissional – a informação é automática e eletronicamente registrada;
- uma base de informação acessível – a informação registrada pelo despachador é mais legível e pode ser transportada de um lugar a outro;
- um plano ótimo refletindo políticas administrativas – as ultrapassagens e cruzamentos consideram prioridades chaves de acordo com políticas de administração;
- uma atitude mais justa com o trabalho do despachador – antes os despachadores eram muito criticados pelos atrasos que aconteciam;
- redução de custo operacional – a redução dos atrasos sofridos pelos trens se traduz direitamente numa redução de custos;
- maior conforto no trabalho por parte da tripulação – a tripulação ao sofrer menos atrasos sente-se mais motivada no trabalho.

Os primeiros modelos de despacho de trens estavam todos baseados em programação matemática convencional e teorias de otimização [OZE94]. Estes modelos apresentam numerosas desvantagens discutidas em [JIA93] e [TSU88]. Entre as mais importantes temos:

- estes modelos não correspondem significativamente ao comportamento da operação dos trens como visto pelos operadores, despachadores e supervisores;
- eles envolvem um grande esforço computacional, o que se torna um impedimento para cumprir com restrições de tempo real impostas em sistemas de despacho integrados – aqueles que envolvem comunicação direta com os maquinistas;

- não refletem a habilidade com a qual o despachador lida com a imprecisão dos dados nem refletem a experiência dele.

Estas desvantagens levaram aos pesquisadores a procurar novas técnicas que pudessem ser aplicadas em lógicas de despacho. Assad [ASS80a] menciona que a quantidade excessiva de trabalho envolvido na lógica de despacho de trens não podia continuar sendo dirigida por uma só unidade. Segundo ele, o trabalho tinha que ser distribuído entre vários agentes. Nesse artigo, também fala-se sobre a aplicação de heurísticas que ajudem na solução de problemas complexos e que, além disso, expressem melhor a maneira e a experiência do despachador. Graças aos avanços nas pesquisas existentes, estas desvantagens começam a ser superadas.

O periódico *Transportation Research Part A: General* editou um número completo dedicado a incorporação da Inteligência Artificial (IA) nos sistemas de transporte. Nele Stephen Ritchie [RIT90], um dos pioneiros na pesquisa de IA, faz uma introdução ao número e ressalta o avanço nas pesquisas de IA graças à flexibilidade que proporciona na representação do conhecimento. Ele também faz menção aos sistemas especialistas (SE), indicando-os como a primeira ferramenta aplicada aos sistemas de transporte e vislumbra uma maior aplicação dela na solução destes problemas. Os SE representam um paradigma de referência para implementar sistemas de controle de tráfego, graças às representações explícitas do conhecimento que facilitam as tarefas de manutenção e monitoração.

Além dos Sistemas Especialistas outras ferramentas da Inteligência Computacional (IC) têm sido aplicadas na solução de problemas de controle do tráfego, dentre elas destacando-se os Sistemas Nebulosos, Inteligência Artificial Distribuída (DAI), Redes Neurais e Algoritmos Genéticos.

No problema de despacho de trens, a decisão que toma o despachador está baseada em atributos imprecisos – e.g. "o trem está *muito* atrasado" – conseqüentemente o processo de tomada de decisões tem também uma natureza imprecisa. A lógica nebulosa e os conjuntos nebulosos são mecanismos convenientes para representar a natureza imprecisa do problema de despacho. Elas constituem uma ferramenta real para atenuar as deficiências apresentadas por métodos da matemáticos convencionais [JIA94].

Para tratar da restrição temporal imposta pelos sistemas de despacho integrados, utiliza-se a Inteligência Artificial Distribuída. A DAI propõe uma arquitetura composta por módulos separados que cooperam para resolver um problema ou para alcançar um objetivo.

Cada módulo é encarregado de resolver uma parte do problema e está provido com pelo menos um sistema baseado em conhecimento. A quantidade e a forma de conhecimento disponível em cada módulo e o grau de certeza nele vai gerar diferentes tipos de arquiteturas de DAI.

A incorporação da IA e da IC com suas ferramentas – lógica nebulosa, sistemas especialistas, inteligência artificial distribuída, redes neurais e algoritmos genéticos - assim como outras ferramentas, proporcionam ao despacho de trens maior flexibilidade, permitindo assim a divisão das tarefas de despacho e uma melhor expressão do conhecimento do despachador.

Existem duas formas de controle considerando à divisão do trabalho envolvido na tarefa de despacho. A primeira delas, e a mais usada, é o Controle de Tráfego Centralizado (CTC). Nestes sistemas, a tomada de decisões e o processamento da informação estão todas centralizadas numa só unidade. Neles os despachadores controlam o movimento dos trens sobre a linha ferroviária, planejam as ultrapassagens e cruzamentos, assumem o controle dos sinais, realizam a compilação e o registro da informação, e se encarregam da comunicação [SAU83]. As primeiras lógicas de despacho forem concebidas nesta estrutura. Exemplos destes sistemas podem ser observados em [PET86], [SZP73], [JOV90], [HO97] e [GOM99]. A performance destes sistemas depende principalmente da capacidade e eficiência da unidade central de controle. Esta estrutura apresenta-se débil quando é implementada como um sistema de despacho integrado, pois este tipo de sistema apresenta restrições de tempo de resposta e o tempo que se requer para que só uma unidade realize a tomada de decisões pode ser elevado.

A outra forma de controle de um sistema de despacho de trens é a de Controle de Tráfego Distribuído (CTD). Sob esta estrutura, o controle do tráfego dos trens e a tomada de decisões está distribuída ao longo das diferentes hierarquias criadas. Estas hierarquias são estipuladas de acordo com as diferentes naturezas e requisitos de tempo impostas. Em alguns casos, a estrutura central será encarregada de lidar com a tomada de decisões estratégicas e com a administração no nível global. As decisões em tempo real serão tomadas pelos controladores locais. Cada um destes controladores modificará a escala dos trens alocados nessa área usando só os recursos locais. As restrições que existem nos CTC são relaxadas nos sistemas distribuídos. Além desta vantagem o CTD é superior ao CTC em

flexibilidade, confiabilidade, clareza, expansibilidade, e imunidade a distúrbios operacionais causados pela atualização ou reconstrução do sistema [VER90]. Na última década foram desenvolvidos alguns CTD com o firme propósito de superar as restrições de tempo impostas pelos sistemas de despacho integrados. Exemplos destes sistemas podem ser observados em [JIA94] e [VER90].

Assad [ASS80a] fornece uma classificação formal dos modelos de suporte à operação ferroviária. Segundo ele, estes modelos podem ser classificados como modelos de otimização (orientados pelo objetivo) ou simulação (orientados pela decisão). A seguir se faz a distinção entre ambos tipos de modelos.

Os modelos de otimização procuram um encaminhamento do tráfego na linha ferroviária com respeito a uma função objetivo que pode estar associada a custos ou atrasos dos trens. O mínimo que os modelos precisam como entrada é o ponto de início e término do percurso de cada trem. Exemplos de modelos de otimização que mantém a velocidade constante podem ser observados em [PET75], [HO97], [JOV91], [CAR95], [CAR94a], [CAR94b], [KRA95], [BRA96] e [NOU97]. Os modelos de otimização que consideram a velocidade como uma variável de decisão não são muito comuns embora apresentem uma melhoria significativa com respeito aos modelos de velocidade constante. Estes modelos, além de reduzir o atraso dos trens, também quantificam e reduzem o consumo de combustível [COR98]. Exemplos destes modelos podem ser observados em [KRA91], [HIG96] e [HIG97].

Os modelos de simulação representam o movimento dos trens na linha ferroviária a partir de um conjunto planos e atividades a serem cumpridas nos pátios e na linha. Deste modo o usuário tem que inserir as atividades dos pátios, o itinerário completo dos trens, e restrições de tráfego entre outros dados. A saída destas simulações pode incluir custos operacionais e informação acerca da distribuição dos tempos dos trens. Baseando-se nestes resultados os usuários podem avaliar as políticas aplicadas com detalhe, ou podem fazer comparações entre diversas alternativas de estratégias de decisão. Estes modelos, dependendo de seu escopo, podem acomodar um grande nível de detalhes. Apresentam um comportamento transitório do sistema, são de uso e aceitação considerável, porém com algumas limitações que variam de acordo com as restrições impostas. Exigências excessivas de dados para utilizar o modelo e dificuldades em manusear tráfegos de alta intensidade,

devido as possibilidades de bloqueio, podem torná–los ineficientes. Allman [ALL66] implementou um dos primeiros simuladores de despacho. Seu primeiro modelo foi usado para realizar uma simulação de dez dias em uma linha ferroviária com 11 pátios e 28 trens. O modelo usava como entrada o itinerário dos trens, planos de atividades nos pátios e informação dos custos operacionais. Petersen e Taylor [PET82] desenvolveram um modelo de simulação baseado no conceito de eventos discretos, visando a movimentação dos trens em uma linha ferroviária. O modelo é aplicável a qualquer linha com segmentos simples, duplos ou múltiplos, conexões completas ou restritas, com ou sem sinalizações intermediárias. Neste modelo também é permitido introduzir diversas estratégias de despacho.

2.3. Modelo Proposto

O objetivo principal do trabalho é o de desenvolver um modelo de linha que seja descrito em um espaço de estados apropriado para resolver o problema de mover trens em uma linha, e uma descrição das relações algébricas que são necessárias para garantir as considerações de factibilidade e segurança.

O modelo proposto está baseado no modelo descrito em [PET82]. Ambos são modelos de simulação e otimização que permitem:

- calcular a performance de cada tipo de trem sobre uma linha;
- avaliar a capacidade de linhas e sistema;
- determinar a performance de linhas em função do tráfego;
- analisar a performance em função da linha e parâmetros de operação;
- avaliar estratégias de atualização física da linha;
- avaliar estratégias de despacho;
- analisar performance de sistemas de ferrovia– terminais–pátios.

O modelo proposto permite trabalhar com linhas ferroviárias que apresentem bifurcações (ramais), ao contrário do modelo proposto em [PET82] que só trabalha com uma rede de segmentos consecutivos.

Ao modelo está agregada uma ferramenta que deverá permitir a inclusão de técnicas de despacho baseadas na lógica nebulosa, incrementando desta maneira a flexibilidade e versatilidade inicial do modelo para refletir a situação real.

Para implementar o modelo, técnicas de análise e projeto orientado a objetos são utilizadas. Graças as características intrínsecas da orientação a objetos, o modelo apresenta a flexibilidade, clareza, expansibilidade e a facilidade para se atualizar ou reconfigurar um dado sistema ferroviário [VER90].

A implementação do modelo, usando a linguagem Java, proporciona a portabilidade para vários ambientes e a sua utilização por terminais remotos via rede de computadores. Além disso, as ferramentas para implementação de interfaces gráficas fornecida pela linguagem Java, proporciona ao usuário um ambiente agradável e amigável de trabalho.

No capítulo seguinte detalha-se o modelo de linha ferroviária proposto em por Petersen e Taylor e o modelo de linha proposto neste trabalho.

2.4. Resumo

Neste capítulo apresentou-se uma visão geral e uma revisão bibliográfica sobre modelos de linha e modelos de despacho de trens.

Modelos de linha têm sido propostos com a finalidade de estimar o desempenho de cada trem causado pela interferência no tráfego em uma linha ferroviária. Estas interferências são dependentes das decisões de despacho, distribuição do tráfego, topologia física da linha ferroviária e comportamento de terminais de carga, descarga e transbordo.

Os modelos de despacho de trens visam otimizar o consumo de combustível, incrementar a capacidade da linha, incrementar a confiança no serviço, etc. Os primeiros modelos de despacho de trens apresentam uma estrutura centralizada, sendo somente um agente o encarregado da tomada de todas as decisões. Restrições de tempo levaram à criação de sistemas de despacho com estrutura distribuída, dividindo a tomada de decisões entre vários agentes.

Os modelos de despacho existentes estão classificados em dois grupos. Os modelos de otimização procuram um encaminhamento do tráfego na linha ferroviária considerando uma função objetivo. Os modelos de simulação representam o movimento dos trens em uma linha, a partir de dados de trens, planos e atividades de linhas e de pátios.

O sistema proposto neste trabalho está baseado no modelo apresentado em [PET82]. O sistema permite considerar linhas ferroviárias com vários ramais, e facilita a inclusão de técnicas de despacho, particularmente aquelas baseadas na lógica nebulosa.

Capítulo 3

Modelo de Linha Proposto

Este capítulo aborda a especificação da estrutura do modelo de linha ferroviária de interesse neste trabalho. Esta especificação é feita através da descrição de seus componentes e da lógica que os relaciona. Estes componentes são descritos textualmente e variáveis a eles associadas para explicitar a lógica do modelo de linha.

Como mencionado anteriormente, o sistema desenvolvido está baseado no modelo de Petersen e Taylor. Este modelo foi modificado para estender suas características iniciais. Na seção 3.1 descreve-se os elementos do modelo original, na seção 3.2, sua lógica e na seção 3.3 descreve-se o modelo proposto.

3.1. Estrutura do Modelo de Petersen e Taylor

Nesta seção apresenta-se os componentes envolvidos no modelo de linha proposto por Petersen e Taylor [PET82]. Para estes componentes definem-se variáveis que são usadas na definição das expressões algébricas que caracterizam o funcionamento do modelo. Estas expressões são descritas na seção 3.2.

3.1.1. Descrição de trens

O modelo original de Petersen e Taylor supõe que existem n trens a serem despachados sobre a linha ferroviária, estes trens estão separados em dois conjuntos disjuntos *OB* e *IB*. Estes dois conjuntos representam trens viajando no sentido oeste – leste (*outbound*) e trens viajando no sentido leste – oeste (*inbound*).

Assume-se que cada um destes trens tem um horário de partida, um horário de chegada desejado, um comprimento, uma velocidade de percurso e uma prioridade. A prioridade de um trem está relacionada com a preferência que este terá quando estiver competindo com outro trem pela ocupação de um mesmo segmento da linha. Quanto maior for este valor, maior será sua preferência.

Tomando como base esta descrição dos trens, associa-se a cada trem i de *OB* ou *IB* os seguintes atributos:

depart(i)	=	horário de partida;
arrive(i)	=	horário de chegada desejado;
speed(i)	=	velocidade de percurso;
length(i)	=	comprimento do trem;
pry(i)	=	prioridade do trem.

3.1.2. Configuração da Linha Ferroviária

Assume-se a linha ferroviária dividida em um número fixo de segmentos. Os segmentos representam a extensão de um trecho entre dois pontos de transferência, desvios, pátios ou estações. Cada um destes segmentos tem um comprimento, um número de vias (ou trechos) paralelas e um número de sinalizações intermediárias. Estes segmentos são enumerados arbitrariamente 1,2,...M no sentido *outbound*. A velocidade máxima permitida nas locações – onde locação simboliza uma via particular em um segmento – não é sempre a mesma. Geralmente, a velocidade permitida na via principal é maior que todas as outras.

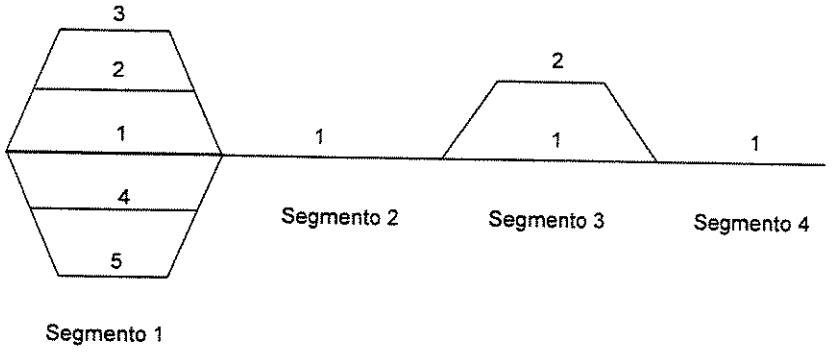


Figura 3.1 Configuração de uma extensão de linha ferroviária

Na Figura 3.1 observa-se uma extensão de linha ferroviária. Nela coexistem 4 segmentos: 1, 2, 3, 4. O segmento 1 tem 5 trechos paralelos, o segmento 2 tem 1 trecho, o segmento 3 tem 2 trechos paralelos e o segmento 4 tem 1 trecho. O trecho 1 do segmento 1 simboliza uma locação, o trecho 2 do segmento 1 simboliza outra locação, o trecho 3 do segmento 1 simboliza outra locação e assim sucessivamente até o trecho 1 do segmento 4 que simboliza a última locação.

A Figura 3.2 mostra que, apesar dos segmentos 1 e 2 serem consecutivos, todos seus trechos não estão conectados fisicamente. Por exemplo, o trecho 1 do segmento 1 não tem conexão física com o trecho 2 do segmento 2. Esta característica, faz necessária o estabelecimento de matrizes de conexões que descrevam os caminhos possíveis.

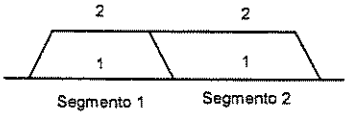


Figura 3.2 Conexões físicas entre segmentos consecutivos

Adicionalmente, assume-se que os trens param a uma determinada distância antes do cruzamento. Além disso, os trens têm que manter uma distância mínima de separação entre eles. Se existir mais de uma sinalização intermediária no segmento, os trens poderão seguir uns aos outros mantendo esta distância mínima.

Considerando esta descrição da linha ferroviária associa-se a ela atributos:

M	=	número de segmentos da linha ferroviária;
dsig	=	distância mínima do cruzamento antes da qual o trem deve parar;
hdist	=	distância mínima de separação que deve existir entre dois trens para garantir segurança no percurso.

Para cada segmento m, definem-se os seguintes atributos:

$dseg(m)$	=	comprimento do segmento;
$ntrack(m)$	=	número de trechos paralelos no segmento;
$u(m)$	=	número de sinalizações intermediárias;

Para cada trecho k de cada segmento m, define-se a seguinte variável:

$v(m,k)$	=	velocidade máxima permitida no trecho k do segmento m.

As matrizes que representam as conexões físicas entre as diversas locações são definidas da seguinte forma:

$KI(m,k,k1)$	=	elemento da matriz de conexão no sentido *inbound*. Assume o valor 1 se um trem viajando no sentido *inbound* pode passar do trecho k do segmento m até o trecho $k1$ do segmento $m-1$. Caso contrário, assume valor 0;
$KO(m,k,k1)$	=	elemento da matriz de conexão no sentido *outbound*. Assume o valor 1 se um trem viajando no sentido *outbound* pode passar do trecho k do segmento m até o trecho $k1$ do segmento $m+1$. Caso contrário, assume valor 0.

3.1.3. Atividades de Manutenção

O modelo permite definir um conjunto de atividades de manutenção preestabelecidas. Cada atividade de manutenção a ser realizada em um trecho de um segmento e terá um horário de início e um horário de término.

As variáveis do modelo relacionadas com as atividades de manutenção são:

ψ	=	conjunto de atividades de manutenção.

Para cada $i \in \psi$ define-se:

$MM(i)$	=	número de segmento no qual será realizada a atividade;
$KM(i)$	=	número do trecho no qual será realizada a atividade;
$z1(i)$	=	hora de início da atividade;
$z2(i)$	=	hora do fim da atividade.

3.1.4. Informação sobre os Tempos

Cada trem i precisa de um tempo $t(i,m,k)$ para atravessar o trecho k do segmento m. Este tempo depende de restrições nas velocidades e podem variar de um trecho para outro. Similarmente, $h(i,m,k)$ é o tempo mínimo de separação que deve existir entre o trem i e outros trens que viajam para garantir a segurança.

Existem também tempos que relacionam os diferentes trens com os cruzamentos entre locações. Quando um trem trafega de um trecho para outro, ele perde um intervalo de tempo (τ_1) no segmento anterior e um intervalo de tempo (τ_2) no segmento seguinte devido à desaceleração e a aceleração do trem. Um trem demanda um intervalo de tempo

(τ_t) para ultrapassar o final de um segmento. Finalmente, define-se um intervalo de tempo τ_d, como sendo o intervalo de tempo mínimo requerido para que o despachador perceba que um trem deixou uma locação e libere as sinalizações para permitir o ingresso de outro trem.

3.1.5. Trens Imaginários

O modelo de Petersen e Taylor contempla a criação de trens imaginários em duas condições.

A primeira condição refere-se a razões de segurança. Requisitos de segurança ditam que só um trem pode ocupar um trecho em um mesmo instante. Isto significa que enquanto o trem está passando de um trecho para outro, ambos trechos devem estar reservados para uso exclusivo desse trem durante um intervalo de tempo adequado, para evitar que ocorram colisões. Os trens só podem seguir uns aos outros se for mantida entre eles uma separação apropriada. Para representar esta condição, sempre que um trem deixa um trecho, um trem imaginário é criado para ocupá-lo durante um intervalo de tempo até tornar-se disponível para o uso de outros trens. No fim do intervalo de tempo o trem imaginário é removido do modelo.

A segunda condição relaciona-se com as atividades de manutenção. Em alguns momentos, trechos da linha ferroviária podem ficar sem condições de uso, seja porque precisam de manutenção ou por algum outro motivo. Isto é representado criando-se um trem imaginário ocupando esse trecho durante o intervalo de tempo em que transcorre a interrupção.

Define-se as seguintes variáveis relacionadas com os trens imaginários:

PI	=	lista de trens imaginários viajando no sentido *inbound*
PO	=	lista de trens imaginários viajando no sentido *outbound*
PM	=	lista de trens imaginários criados por razões de manutenção/interrupção da linha.

Anteriormente foram definidos dois conjuntos de trens na linha ferroviária OB e IB. Estes conjuntos junto com os definidos para os trens imaginários formam o conjunto de trens η que representa todos os trens na linha ferroviária:

$$\eta = OB \cup IB \cup PO \cup PI \cup PM$$

3.1.6. Lógica do Modelo

O modelo de Petersen e Taylor é um modelo de simulação a eventos discretos que avança no tempo de acordo com a ocorrência de eventos. Neste modelo, um evento corresponde ao início e término de uma atividade de um trem. Se o evento ocorrido for gerado por um trem imaginário ou um trem chegando a seu destino, o mesmo é retirado do modelo da linha ferroviária, avançando o processo de simulação para o evento seguinte. Quando o evento não for gerado nem por um trem imaginário nem por um trem chegando a seu destino, a rotina do despacho decide se o trem pode avançar para uma determinada locação ou se o trem deve ficar parado na sua locação atual.

O estado do trem é atualizado para refletir sua nova condição e o momento no qual sua nova atividade terminará. Se o trem ocupa uma nova locação, um trem imaginário é criado para ocupar a locação anterior até que esta possa ser usada por um outro trem. Em seguida, a simulação avança para o evento seguinte.

O estado do sistema é completamente especificado conhecendo-se o conteúdo de cada trem em η, junto com as seguintes variáveis:

$x(i)$	=	hora na qual o trem i completará a sua atividade atual;
$A(i)$	=	número do segmento onde está posicionado o trem i;
$B(i)$	=	número de trecho onde está posicionado o trem i;
$D(i)$	=	1 se o trem está parado em um trecho ou 0 em caso contrário.

onde $i \in \eta$.

Na próxima seção descreve-se a estrutura algébrica do modelo.

3.2. Especificação Algébrica da Lógica do Modelo de Petersen e Taylor

Nesta seção detalha-se as relações analíticas que caracterizam o modelo de Petersen e Taylor. Estas relações são especificadas fazendo-se uso das variáveis e parâmetros definidos na seção anterior.

O sistema é inicializado, criando-se os conjuntos IB e OB com os trens a serem despachados na linha ferroviária. As variáveis que representam o estado do sistema são inicializadas usando as equações 3.1 a 3.4 onde $i \in \eta$.

$$x(i) = depart(i) \tag{3.1}$$

$$A(i) = \begin{cases} 1, & \text{se } i \in OB \\ M, & \text{se } i \in IB \end{cases} \tag{3.2}$$

$$B(i) = 1 \tag{3.3}$$

$$D(i) = 1 \tag{3.4}$$

Após a inicialização, as seguintes atividades devem ser executadas.

A primeira etapa é determinar qual é o próximo evento a ser considerado; isto é, aquele que acontece no momento $x(l)$ e envolve o trem l (conforme Eq. 3.5) onde:

$$x(l) = min_{\,i \in \eta}\{x(i)\} \tag{3.5}$$

Se o trem l chegou ao seu destino final ou l é um trem imaginário, então o mesmo é removido do sistema usando Eq. 3.6, e o próximo evento deve mais uma vez ser escolhido usando Eq. 3.5.

$$\begin{aligned} &l \in PO \Rightarrow PO = PO - \{l\} \\ &l \in PI \Rightarrow PI = PI - \{l\} \\ &l \in PM \Rightarrow PM = PM - \{l\} \\ &l \in OB \wedge A(i) = M \Rightarrow OB = OB - \{l\} \\ &l \in IB \wedge A(i) = 1 \Rightarrow IB = IB - \{l\} \end{aligned} \tag{3.6}$$

A etapa seguinte consiste em verificar se o sistema entrou em um período no qual é necessário interromper um trecho durante um intervalo de tempo. As relações algébricas que especificam esta atividade são descritas na seção 3.2.1.

As próximas etapas estão relacionados com a Rotina de Despacho. Esta se encarrega de solucionar conflitos entre trens e de encontrar um trecho para onde deslocar o trem sendo analisado (trem l). As relações algébricas dessas atividades e uma explicação detalhada das mesmas são descritas na seção 3.2.2.

Se a rotina de despacho encontra uma locação para deslocar o trem l, este trem pode continuar com o seu percurso. Caso contrário, o trem é atrasado na sua posição atual. Isto é, o trem mantém sua posição corrente e o valor de $x(l)$ é alterado.

Após qualquer uma destas atividades, o estado do sistema é atualizado e o modelo avança para o próximo evento. As expressões algébricas destas atividades são descritas nas seções 3.2.3 e 3.2.4.

3.2.1. Atividades de Manutenção

Para saber se, em um dado instante, um trecho estará inacessível, é necessário comparar o instante de tempo no qual ocorrerá o próximo evento – $x(l)$ – com o intervalo de tempo durante no qual o trecho estará interrompido, isto é:

$$z1(i) \leq x(l) \leq z2(i), i \in \psi \tag{3.7}$$

Para todo $i \in \psi$ que satisfaz (3.7) criam-se trens imaginários. As seguintes relações representam a criação de um trem imaginário *pt*.

$$x(pt) = z2(i) \tag{3.8}$$

$$A(pt) = MM(i) \tag{3.9}$$

$$B(pt) = KM(i) \tag{3.10}$$

$$D(pt) = 1 \tag{3.11}$$

O trem imaginário *pt* criado é inserido na lista de trens imaginários:

$$PM = PM \cup \{pt\} \tag{3.12}$$

3.2.2. Rotina de Despacho

O despacho decide se um trem pode continuar seu trajeto ou deve ser atrasado na sua atual locação. Para que um trem possa seguir o seu trajeto, deve-se encontrar uma locação *k1* para onde possa ser deslocado sem ocasionar futuros bloqueios na linha ferroviária. Para tal, é necessário determinar o conjunto K de locações que estão disponíveis para o trem *l* ocupar. A seção 3.2.2.1 mostra as relações que caracterizam este conjunto.

O fato de uma locação estar disponível não implica que a mesma possa ser ocupada pelo trem *l*. Primeiro deve-se garantir que o deslocamento do trem *l* até essa locação não ocasione bloqueio na linha ferroviária. Entende-se como bloqueio na linha ferroviária a situação na qual, nenhum trem na linha pode movimentar sem que outros trens se movimentem no sentido oposto.

Este problema tem sido objeto de estudo, as soluções encontradas são computacionalmente custosas. Neste trabalho, desenvolveu-se um algoritmo simples e computacionalmente eficiente, que permite evitar o bloqueio da linha [RON00a]. Os detalhes sobre este algoritmo são apresentados na seção 3.4.3.

A essência da rotina de despacho pode ser resumida pelo seguinte algoritmo:

1. Seja *m1* o segmento para o qual tem-se que deslocar o trem *l* (Eq. 3.13)

$$m1 = \begin{cases} A(l)+1, & \text{se } l \in OB \\ A(l)-1, & \text{se } l \in IB \end{cases} \tag{3.13}$$

2. Resolver conflitos entre trens concorrendo pelo uso do segmento *m1*.
3. Obter o conjunto K de locações que podem ser ocupadas pelo trem *l*.
4. Fazer *k1=0*.
5. Fazer enquanto $k1 = 0$ e existir um $i \in K$
   - 5.1 $k1 = \{ i,\ i \in K\}$.
   - 5.2. Se o deslocamento do trem *l* ao trecho *k1* ocasiona bloqueio na linha de trens:
     - 5.2.1. $K = K - \{k1\}$
     - 5.2.2. $k1 = 0$
6. Se $k1 = 0$, o trem *l* é atrasado na sua locação atual.
7. Caso contrário, o trem *l* é despachado para a locação *k1*.

A resolução de conflitos entre trens e os passos necessários para que um trem continue seu trajeto ou seja atrasado são explicados nas seções 3.2.2.2, 3.2.3 e 3.2.4 respectivamente.

3.2.2.1. Locações Disponíveis

O trem *l* pode ser deslocado a uma locação *k1* do segmento *m1* se as seguintes condições são satisfeitas:

- as condições físicas o permitem;
- a locação está vazia; ou
- o segmento *m1* tem mais de uma sinalização intermediária que permita que os trens mantenham uma separação mínima estabelecida e:
  - a locação não está ocupada por trens viajando no sentido oposto ao trem *l*;
  - o número de trens viajando no mesmo sentido que o trem *l*, alocados no segmento *m1*, é menor que o número de sinais intermediárias que tem o segmento.

Estas condições são traduzidas no seguinte:

$$\begin{aligned} K(m,k,l,S,Kr) = \{ & k1 \mid KK(m,k,k1) = 1 \text{ e } (k1 \notin Kr) \text{ e} \\ & U_{i \in S}[(B(i) \neq k1) \text{ ou} \\ & \quad [(u(m1) > 1) \text{ e } (num(Q) <= u(m1))] \\ & \} \end{aligned} \tag{3.14}$$

Onde:

$$KK(m,k1,k2) = \begin{cases} KO(m,k1,k2), & l \in OB \\ KI(m,k1,k2), & l \in IB \end{cases} \tag{3.15}$$

é a matriz de conexão no sentido de viagem;

$$S = \{ i \mid A(i) = m1, \ \forall i \in \eta \} \tag{3.16}$$

é o conjunto de trens que ocupam o segmento *m1*;

$$Q = \{ i \mid B(i) = k1, \ \ i \in S \cap G(l) \} \tag{3.17}$$

é o conjunto de trens que ocupam a locação *k1* no sentido de viagem do trem *l*;

$$G(l) = \begin{cases} OB, & l \in OB \\ IB, & l \in IB \end{cases} \tag{3.18}$$

Kr= lista de locações que não podem ser ocupadas por estarem reservadas para outros trens.

3.2.2.2. Resolução de Conflitos entre Trens

Para que a rotina de despacho escolha uma locação *k1* para ser ocupada pelo trem *l*, é evidente que ela tem que conhecer quais locações não podem ser escolhidas. Obviamente, parte das locações que não podem ser escolhidas são aquelas que estão ocupadas e cujo segmento só tem uma sinalização intermediária e outras são aquelas locações que não são fisicamente acessíveis a partir da locação atual do trem *l*. Entretanto, existem outras locações que não podem ser escolhidas porque estão reservadas para outros trens. Esta seção trata da reserva dessas locações.

É comum encontrar conflitos entre trens que pretendem ocupar um mesmo segmento ao mesmo tempo. Nestes casos, a rotina de despacho deve definir as ações, em concordância com os objetivos que pretendem ser alcançados.

Chama-se trens concorrentes o conjunto de trens que estão competindo pela ocupação do segmento *m1* durante um mesmo intervalo de tempo. Para saber quais são os trens concorrentes, deve-se determinar o intervalo de tempo no qual existe competição. Para isto, determina-se o tempo *delta* (Eq. 3.19), tempo este necessário para o trem *l* percorrer os dois segmentos seguintes ao longo dos trechos mais rápidos, isto é:

$$delta = t(l,m1,1) + t(l,m2,1) \tag{3.19}$$

sendo *m2* o segmento próximo ao segmento *m1* considerando o sentido de viagem do trem *l*.

Portanto, o intervalo de tempo durante o qual ocorrerá a competição, será definido entre : $[x(l)\ ,x(l) + delta]$.

O conjunto de trens concorrentes do segmento *m1*, na faixa de tempo requisitado pelo trem *l*, é definido por:

$$\Phi(m1, x(l), delta) = \{i \mid x(l) \leq y_i(m1) \leq x(l) + delta, i \in OB \cup IB\} \qquad (3.20)$$

onde $y_i(m1)$ é a hora na qual o trem i chegará ao segmento $m1$.

Os elementos Φ são, portanto, aqueles que competem com o trem l pela ocupação do segmento $m1$. Já que nem todos podem ocupá-lo no mesmo instante, reserva-se locações para alguns trens. A escolha destes trens depende dos objetivos do sistema.

3.2.2.3. Seleção de Trens

Quando vários trens estão concorrendo pela ocupação de um mesmo segmento, deve-se selecionar quais destes trens terão uma locação reservada.

Conforme mencionado, esta seleção dependerá do objetivo especificado que se pretenda alcançar. Por exemplo, pode-se dar sempre preferência aos trens com maior prioridade, escolher aqueles que minimizem os atrasos no sistema, etc.

Como exemplo de um processo de seleção baseado em prioridade estabelecida para os diferentes trens, calcula-se, para cada trem concorrente, a hora em que este percorrerá o segmento m e subtrai-se desse valor uma constante multiplicada pelo valor da sua prioridade. O trem selecionado, *sel_train*, será aquele que corresponda ao mínimo valor calculado:

$$sel\_train = \min_{i \in \Phi} \{i \mid y_i(m) + t(i, m, 1) - const * prty(i)\} \qquad (3.21)$$

onde a expressão $y_i(m) + t(i,m,1)$ fornece a hora em que o trem i atravessará o segmento m (hora em que chega no segmento mais o tempo que leva para atravessá-lo). Quanto maior a prioridade de um trem concorrente, maior será a quantidade a ser diminuída, aumentando a possibilidade de ser escolhido.

Se o trem selecionado não for o trem l, procede-se escolhendo uma locação a ser reservada kr. A locação (kr) é inserida no conjunto de locações reservadas Kr (Eq. 3.22) e o trem selecionado (*sel_train*) é inserido no conjunto de trens reservando locações (Eq. 3.23). Após reservar locação para o trem selecionado, procura-se no

conjunto de trens concorrentes um outro trem a selecionar e repete-se o processo descrito.

$$Kr = Kr \cup \{ kr \} \tag{3.22}$$

$$TRES = TRES \cup \{sel\_train\} \tag{3.23}$$

No caso do trem selecionado for o trem *l*, a seleção de trens para lhes reservar locações termina.

3.2.3. Movimentação de Trens

Nesta seção, são detalhadas as ações tomadas quando encontra-se uma locação *k1* para o trem *l* ocupar.

Por razões de segurança uma locação não pode ser ocupada imediatamente após ser liberada. É preciso esperar um tempo apropriado para que o trem *l* avance e não ocorram colisões. Por isto, é necessário criar um trem imaginário e alocá-lo no trecho ocupado pelo trem *l*. Assim, seja *pt* o trem imaginário a ser criado. As relações que envolvem sua criação são as seguintes:

$$A(pt) = m \tag{3.24}$$

$$B(pt) = k \tag{3.25}$$

$$x(pt) = x(l) + \tau_s \tag{3.26}$$

$$P = \begin{cases} PO, & l \in OB \\ PI, & l \in IB \end{cases} \tag{3.27}$$

$$P = P \cup \{pt\} \tag{3.28}$$

onde:

τ_s = intervalo de tempo em que o trem imaginário permanecerá na linha.

Ao ser movimentado para a locação *k1* o trem *l* provoca mudança de estado da linha. Esta mudança é sumarizada pelas equações 3.29 a 3.33 abaixo:

$$m = A(l) \tag{3.29}$$

$$k = B(l) \tag{3.30}$$

$$A(l) = m1 \tag{3.31}$$

$$B(l) = k1 \tag{3.32}$$

$$x(l) = x(l) + t(l, m1, k1) + \tau_1 + \tau_2 \tag{3.33}$$

onde:

$t(l, m1, k1)$ = tempo que o trem l leva para atravessar o trecho $k1$ do segmento $m1$;

τ_1 = tempo perdido devido a aceleração e desaceleração no segmento $A(l)$;

τ_2 = tempo perdido devido a aceleração e desaceleração no segmento $m1$;

Depois de alocar o trem l em $k1$, deve-se analisar o estado do segmento onde está ($m1$) e o segmento onde esteve (m). Se na nova locação do trem l existirem outros trens, estes poderão eventualmente ser atrasados dependendo da hora do término de suas atividades. Se na locação anterior existirem outros trens, estes também podem ser eventualmente atrasados. A existência de trens nas locações anterior e posterior do trem l, significa que os segmentos dessas locações tem mais de uma sinalização intermediária. Após a consumação dos eventuais atrasos, o sistema encontra o próximo evento a considerar e todo o processo é repetido.

3.2.3.1. Atraso de Trens na Nova Locação

Se na locação onde foi alocado o trem ainda existirem outros trens, o mesmo deverá ser atrasado, para manter os requisitos de segurança.

O trem l pode ser atrasado ou não, considerando a hora de seu próximo evento e a hora na qual o último trem sairá da locação. Seja *last* o trem que sairá por último da locação:

$$x(last) = \{ max_i\, x(i) \mid A(i) = m1 \ e\ i \in \eta \} \tag{3.34}$$

Se :

$$x(l) < x(last) + h(l,m1,k1) \tag{3.35}$$

então:

$$x(l) = x(last) + h(l,m1,k1) \tag{3.36}$$

As expressões 3.33 e 3.34 significam que se a hora na qual o trem *l* sairá, é menor que a hora de partida o trem *last* acrescentado do tempo que é necessário para manter uma distância segura, então é necessário atrasar o trem *l* até esse horário.

3.2.3.2. Atraso de Trens na Antiga Locação

Se na locação anterior percorrida pelo trem *l* existirem mais trens, estes podem ser eventualmente atrasados. O algoritmo que contempla estes possíveis atrasos é o seguinte:

1. Construir o conjunto *DT*, contendo os trens que estão na locação ocupada anteriormente pelo trem *l*.

$$DT = \{ i \mid m = A(i) \, e \, k = B(i) \, , i \in OB \cup IB \} \tag{3.37}$$

2. Fazer $xbase = x(l)$
3. Para cada trem $i \in DT$

   3.1 Fazer:

$$next = min_{\, i \in DT} \{ x(i) \} \tag{3.38}$$

   3.2 Se a condição é cumprida:

$$x(next) < xbase + h(next,m,k) \tag{3.39}$$

   então:

$$x(next) = xbase + h(next,m,k) \tag{3.40}$$

$$D(next) = 1 \tag{3.41}$$

$$xbase = x(next) \tag{3.42}$$

$$DT = DT - \{next\} \tag{3.43}$$

   3.3 Senão, terminar.

3.2.4. Atraso do Trem

Não sendo encontrada uma locação *k1* para onde o trem *l* possa ser despachado, este deve ficar parado na sua posição atual durante um certo tempo. Se nessa locação houver mais trens, então estes também serão atrasados.

O tempo de atraso que o trem *l* sofrerá, caso seu deslocamento produza um bloqueio na linha, dependerá dos trens que estão reservando ou ocupando locações acessíveis ao trem.

Os trens que estão reservando locações são aqueles contidos no conjunto *TRES* conforme a seção 3.2.2.3.

Os trens que estão ocupando locações fisicamente acessíveis ao trem *l* podem ser determinados usando-se o seguinte:

$$TOCU = \{\, i \mid [i \in OB \ \mathbf{e}\ A(i) = m1 \ \mathbf{e}\ KO(m,k,B(i)) = 1] \ \mathbf{ou}$$

$$[i \in IB \ \mathbf{e}\ A(i) = m1 \ \mathbf{e}\ KI(m,k,B(i) = 1] \,\} \qquad (3.44)$$

Dependendo da existência ou não destes trens, existem quatro possíveis caminhos para determinar o tempo de atraso:

3.2.4.1. Não Existem Trens Reservando ou Ocupando Locações Acessíveis

Quando não existem trens reservando ou ocupando locações no segmento *m1*, o tempo que o trem *l* ficará parado dependerá dos trens que estão se movimentando no sentido contrário a dele. Portanto, é necessário encontrar o primeiro trem viajando no sentido oposto do trem *l*. Se este trem pode ser despachado – ou seja, é possível encontrar uma locação para que possa ser despachado sem ocasionar bloqueio na linha – então o trem *l* será atrasado até que o evento desse trem seja cumprido.

Se o primeiro trem encontrado não pode ser despachado, procura-se o trem seguinte.

3.2.4.2. Existem Trens Ocupando e Reservando Locações Acessíveis

Quando existem trens ocupando e reservando locações que podem ser ocupadas pelo trem *l*, o trem *l* deve ser atrasado até que ocorra a primeira liberação. Quer dizer, obtém-se a menor hora na qual os trens que estão reservando locações finalizaram sua atual atividade e a menor hora na qual trens que estão ocupando locações vão desocupá-las. Escolhe-se a menor destas duas horas e atrasa-se o trem *l* até esse momento.

3.2.4.3. Existem Trens Ocupando Locações

Quando existem somente trens ocupando locações que são acessíveis ao trem *l* e não existem trens reservando locações, o trem *l* deve ser atrasado até o instante em que um destes trens seja despachado.

3.2.4.4. Existem Trens Reservando Locações

Quando existem somente trens reservando locações, o trem *l* ficará parado até que o primeiro deles complete a sua atividade.

Depois de realizado o atraso do trem *l*, tem-se que verificar a existência de outros trens na mesma locação que possam sofrer atraso devido ao atraso sofrido pelo trem *l*. O processo para atrasar os outros trens é similar ao usado na seção 3.2.3.2. Depois de atrasar o trem *l*, o sistema determina qual é o próximo evento a ocorrer e todo o processo é repetido.

3.3. Estrutura do Modelo de Linha Proposto

No modelo original de Petersen e Taylor foram feitas algumas modificações visando estender sua capacidade e simplificar alguns aspectos. Estas modificações foram feitas com base no conhecimento e no entendimento das operações observadas em algumas empresas ferroviárias.

O modelo proposto permite a simulação não só de uma linha mas também de uma malha ferroviária. Esta característica do modelo faz que o sistema implementado simule situações mais realistas. Embora algumas empresas ferroviárias possam ser responsáveis só

por uma linha ferroviária, a maioria delas lida direta ou indiretamente com interferências de trens que vêm de outras linhas.

A rota de um trem não começa, necessariamente, no início de uma linha ferroviária ou termina no final dela, como supõe o modelo de Petersen e Taylor. Podem existir pátios ao longo da linha na qual os trens possam manobrar para prosseguir para outros pontos. Além disso, dada a necessidade do modelo de considerar malhas ferroviárias, não é possível designar uma rota de um trem conforme o modelo original. Dadas estas situações, considerou-se necessário mecanismos para a criação de modelos de rotas (ou percursos).

Os trens que circulam em uma ferrovia possuem algumas características comuns. Estas características que compartilham permitem criar modelos (classes) de trens. Exemplos de modelos de trem incluem: trem de minério, trem cargueiro, trem de passageiros, etc.

Alguns trens podem realizar atividades tanto em terminais quanto em pátios. Para representar estas operações criaram-se modelos de escalas.

Em planejamento de circulação é freqüente a necessidade de se avaliar efeitos de perturbações. Portanto, o modelo proposto incorpora a possibilidade de se considerar perturbações nos trens.

As melhorias acima mencionadas são detalhadas nas seções seguintes.

3.3.1. Estrutura da Malha Ferroviária

Uma malha ferroviária pode ser vista como uma lista de segmentos relacionados entre si. Cada segmento está descrito pelos atributos designados no modelo original (número de trechos, comprimento, etc.).

O modelo de Petersen e Taylor supõe que o segmento *m* só pode estar conectado a um único segmento *m1*. Por conseguinte a matriz de conexão original *K(m,k, k1)* só estabelece três argumentos:

m: segmento inicial;

k: trecho do segmento inicial; e

k1: trecho do segmento final.

Dado que numa malha ferroviária um segmento pode estar conectado a mais de um segmento, a matriz de conexão física passa a quatro argumentos. Portanto, as matrizes de conexões físicas entre locações são definidas da seguinte forma:

KI(m,k,m1,k1)	=	elemento da matriz de conexão no sentido *inbound*. Assume o valor de 1 se um trem viajando no sentido *inbound* pode passar do trecho *k* do segmento *m* até o trecho *k1* do segmento *m1*. Caso contrário, assume valor 0;
KO(m,k,m1,k1)	=	elemento da matriz de conexão no sentido *outbound*. Assume o valor de 1 se um trem viajando no sentido *outbound* pode passar do trecho *k* do segmento *m* até o trecho *k1* do segmento *m1*. Caso contrário, assume valor 0.

3.3.2. Modelos de Percurso

Os percursos (rotas) de trens podem ser diferentes entre si, ao invés de serem os mesmos como no modelo original. Por isso define-se modelos de percursos. Um modelo de percurso pode ser seguido por um ou mais trens despachados.

O modelo de percurso está definido por um código, pelo número de segmentos e pela lista de segmentos. As seguintes variáveis são definidas:

journey(i)	=	código do modelo de percurso
nsegs(i)	=	número de segmentos que o compõem o percurso;
JP(i, 1..nsegs(i))	=	lista de segmentos que compõem o percurso.

3.3.3. Modelos de Escalas

Os trens que circulam em uma linha ferroviária têm, em geral, várias atividades a cumprir. Por exemplo: receber e deixar passageiros nas estações, carregar ou descarregar produtos nos pátios, etc. Estas atividades têm um tempo de duração médio e pode sofrer um desvio ao serem realizadas em um determinado segmento. Dado que um trem pode fazer mais de uma escala, estas foram agrupadas em modelos de escalas.

Um modelo de escala tem um código que o identifica e agrupa um conjunto de paradas. Cada parada está caracterizada pela locação que ocorre, o tempo que dura e o eventual desvio neste tempo.

Definem-se as seguintes variáveis:

Para cada modelo de escalas *j*:

nstops(j)	=	número de paradas que compõem o modelo *j*;
stopcode(j)	=	código do modelo de escalas.

Para cada parada *i* de cada modelo *j*

stopseg(j,i)	=	segmento correspondente à parada *i* do modelo *j;*
stoptime(j,i)	=	duração da parada *i* do modelo *j;*
stopdesv(j,i)	=	desvio que pode sofrer a duração da parada *i* do modelo *j*.

3.3.4. Modelos de Trens

Um modelo de trem pode ser considerado como um trem munido com os seguintes atributos: comprimento, número de vagões com uma certa capacidade e comprimento e tempo de percurso em cada segmento que compõe a malha ferroviária.

O tempo de percurso dos diferentes modelos de trens depende da carga (peso) que transportam. Às vezes é necessário considerar trens mais leves ou mais pesados que outros. O modelo proposto contempla a possibilidade de relacionar tempos de dois modelos de trens via fatores multiplicativos. Esta relação cria dependência entre os modelos.

As variáveis necessárias para definir um modelo de trem *j* são as seguintes:

codepattern(j)	=	código do modelo *j*
trainPrty(j)	=	prioridade do modelo de trem
trainLength(j)	=	comprimento do modelo de trem
engine(j)	=	número de locomotivas
typeEngine(j)	=	tipo de locomotiva
carriage(j)	=	número de vagões
carriageCapacity(j)	=	capacidade do vagão
lengthCapacity(j)	=	comprimento do vagão
factor(j)	=	fator de ajuste do tempo de percurso do modelo de trem
trainPatternDependency(j)	=	modelo do trem do qual depende para calcular seus tempos de percurso
trainDescription(j)	=	descrição do trem.
locationtime(j, m, k)	=	tempo de percurso do modelo trem *j* no trecho *k* do segmento *m*

3.3.5. Trens a serem despachados

O modelo original supõe trens despachados somente em um sentido ou no sentido reverso e insere o trem no conjunto *OB* ou *IB*. O modelo proposto considera inicialmente os trens viajando em um sentido e quando estes terminam de percorrer seu trajeto e realizar suas atividades nos pátios ou terminais, considera os trens viajando no sentido reverso (retornando ao seu ponto de origem).

Além disso, quando um trem percorre um trajeto em um sentido ele pode levar uma carga e no sentido oposto pode transportar ou a mesma carga ou uma carga diferente. Também quando em um sentido pode fazer escalas em certos pátios e no sentido oposto fazer escalas em pátios diferentes. O modelo proposto designa dois modelos de trens, dois modelos de escalas e um modelo de percurso para os trens.

O modelo proposto também considera horários de partida perturbados aleatoriamente.

É freqüente se despachar um trem B após um intervalo de tempo que um trem A foi despachado. Dado que a hora de partida do trem A pode não ser determinística (e.g. pode sofrer atrasos), o trem B deve manter uma relação com o trem A para que se saiba quando pode ser despachado. O modelo proposto permite manter esta relação.

Sendo assim, as variáveis associadas com um trem a ser despachado são as seguintes:

traincode (j)	=	código do trem
trainDependency(j)	=	código do trem do qual depende para partir
depart(j)	=	horário de partida ou tempo depois do qual partirá
depart_desv(j)	=	desvio na hora de partida
in_pattern(j)	=	código do modelo de trem quando viaja no sentido *inbound*
out_pattern(j)	=	código do modelo de trem quando viaja no sentido *outbound*
in_stoppat(j)	=	código do modelo de escalas que realiza o trem quando viaja no sentido *inbound*
out_stoppat(j)	=	código do modelo de escalas que realiza o trem quando viaja no sentido *outbound*
journey_code(j)	=	código do modelo de percurso
init_ndir(j)	=	sentido inicial de viagem do trem *j*
ndir(j)	=	sentido de viagem durante a simulação

3.3.6. Trens Imaginários

No modelo original os trens imaginários criados são inseridos em três conjuntos: *PO*, *PI* e *PM*. Os dois primeiros conjuntos contêm trens criados por razões de segurança, enquanto o outro contém os trens criados por razões de manutenção.

Não é necessário manter tantos conjuntos de trens imaginários, nem muito menos definir um sentido de viagem aos trens imaginários criados por razões de segurança. Considera-se conveniente unificar os três conjuntos e manter um único conjunto de trens imaginários *P*.

Para distinguir os trens imaginários criados por razões de manutenção dos trens imaginários restantes designa-se, para cada trem imaginário i de *P*, as seguintes variáveis:

safety(i)	=	assume valor 1 se foi criado por razões de segurança e 0 no caso contrário.
pcode(i)	=	código do trem imaginário

3.3.7. Atividades de Manutenção

O modelo original assume que uma atividade de manutenção inicia-se em um horário determinado e termina em um horário fixado. Dado que estas atividades também podem estar expostas a perturbações, define-se o seguinte para cada $i \in \psi$:

MM(i)	=	número do segmento no qual será realizada a atividade;
KM(i)	=	número do trecho no qual será realizada a atividade;
z1(i)	=	hora de início da atividade;
z1d(i)	=	desvio na hora de início da atividade
z2(i)	=	duração da atividade
z2d(i)	=	desvio que pode sofrer a duração da atividade

3.3.8. Perturbações que podem sofrer os trens

O modelo pode considerar que um trem circulando na linha pode sofrer perturbações. Pode-se querer simular a perturbação de um trem em um momento determinado durante um intervalo de tempo. Para tal, define-se o conjunto λ de trens que sofrem perturbações e para cada trem $i \in \lambda$ define-se:

pcode(i)	=	código do trem a ser perturbado
p1(i)	=	hora de início da perturbação;
p1d(i)	=	desvio na hora de início da perturbação
p2(i)	=	duração da perturbação
p2d(i)	=	desvio que pode sofrer a duração da perturbação
pr(i)	=	probabilidade de ocorrência da perturbação

3.4. Lógica do Modelo Proposto

A base da lógica do modelo proposto segue aquela do modelo de Petersen e Taylor. Isto é, o modelo mantém a essência do comportamento do modelo original:

- inicializa variáveis (estado do sistema);
- procura o próximo trem a ser despachado (próximo evento a ocorrer);
- procura um conjunto de locações para despachar o trem;
- verifica se existe uma locação para ser ocupada pelo trem sem causar bloqueio na linha;
- se existir tal locação o trem é despachado e o modelo avança para o próximo evento;
- se não existir locação para o trem ocupar, este é atrasado e o modelo avança para o próximo evento.

Portanto, considera-se desnecessário especificar estes em detalhes novamente. Cabe sim, ressaltar alguns aspectos importantes.

3.4.1. Eventos do Modelo

O modelo proposto, tal como o original, é um modelo a eventos discretos. No modelo original, considera-se como evento o término da atividade de um trem, sendo o tempo atualizado conforme as atividades.

O modelo proposto contempla a existência de um outro evento: a perturbação que pode sofrer um trem. Na escolha do próximo evento a ocorrer no modelo também considera-se a ocorrência de perturbações.

3.4.2. Funções de Otimização do Despacho

Quando vários trens estão concorrendo pela ocupação de um segmento, deve-se selecionar os trens que terão locações reservadas. Esta seleção dependerá do objetivo que se pretende alcançar. Por exemplo, pode ser o caso de sempre dar preferência a trens com maior prioridade ou escolher a movimentação que reduz ao máximo os atrasos no sistema, etc.

As otimizações do despacho podem ser realizadas usando tanto algoritmos convencionais quanto de inteligência computacional, principalmente aqueles baseados em lógica nebulosa.

O sistema implementado fornece a possibilidade de se definir funções de otimização. Isto é, permite especificar a função objetivo desejada através de uma linguagem que suporta a definição tanto de algoritmos matemáticos usuais quanto de algoritmos baseados em regras e conjuntos nebulosos.

Esta linguagem também fornece a capacidade de obter informação relacionada com os trens concorrentes. Pode-se conhecer quantos trens estão concorrendo pela ocupação de um trecho, qual é a prioridade de um trem, qual é o seu atraso, a que horas cruzará um determinado segmento e com quantos trens se cruzará. A linguagem está descrita na forma B.N.F (Backus Normal Form) no Apêndice.

Na seção 3.2.2.3, apresentou-se um exemplo simples de seleção de um trem, baseado na sua prioridade e no instante que ocupará um segmento. Esta função é descrita usando a linguagem definida a seguir:

```
double dum;
RealTrain AUX_TRAIN;
int const;
double small, temp;
small = 99999;
int size;
size = TRAINS.SIZE;
const = 1000;
for i =1:size
{
    AUX_TRAIN = TRAINS[i];
    dum=AUX_TRAIN.TIMETOCOMPLETE(SEGMENT)+const*AUX_TRAIN.PRIORITY;
```

```
    if(small > dum){
            small = dum;
            SELECTED_TRAIN = AUX_TRAIN;
    }
}
```

No caso de dar preferência ao trem com maior atraso, a função de seleção é descrita da forma seguinte:

```
double dum;
RealTrain AUX_TRAIN;
int const;
double big;
big = -1.0;
int size;
size = TRAINS.SIZE;
const = 2;
for i =1:size
{
    AUX_TRAIN = TRAINS[i];
    dum = AUX_TRAIN.DELAY;
    if(dum >= big){
            big = dum;
            SELECTED_TRAIN = AUX_TRAIN;
    }
}
```

Os dois exemplos de estratégia de despacho apresentados acima são exemplos de algoritmos convencionais. O exemplo seguinte, mostra uma estratégia baseada na lógica nebulosa. Neste exemplo é fornecida uma prioridade ao trem de acordo com seu atraso. Os conjuntos nebulosos do atraso e a prioridade são apresentados na Figura 3.3 e na Figura 3.4, e a estratégia de despacho escrita na linguagem é mostrada a seguir:

```
double big;
double dum;
big = 0.0;
RealTrain AUX_TRAIN;

Atraso = (Pequeno,  Medio ,  Alto);
Prioridade = (Baixa,  Media ,  Alta);

Atraso.Universe = [780,0,1560];
Prioridade.Universe = [50,0,100];
```

```
Atraso.Pequeno = trapese(0,0,240,480);
Atraso.Medio = triang(360,720,1080);
Atraso.Alto = trapese(960,1200,1560,1560);

Prioridade.Baixa = trapese(0,0,20,40);
Prioridade.Media = triang(30,50,70);
Prioridade.Alta = trapese(60,80,100,100);

if X is Atraso.Pequeno then  Z is Prioridade.Baixa
if X is Atraso.Medio then  Z is Prioridade.Media
if X is Atraso.Alto then  Z is Prioridade.Alta

size = TRAINS.SIZE;

for i =1:size
{
    AUX_TRAIN = TRAINS[i];
    X = AUX_TRAIN.DELAY;
    dum = run(0,2);
    if(dum >= big){
            big = dum;
            SELECTED_TRAIN = AUX_TRAIN;
    }
}
```

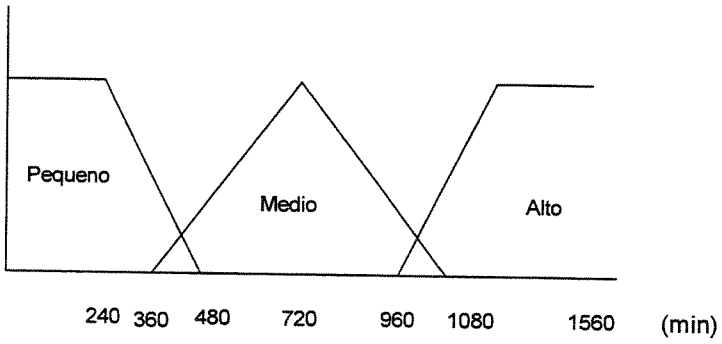


Figura 3.3 Conjunto nebuloso do atraso

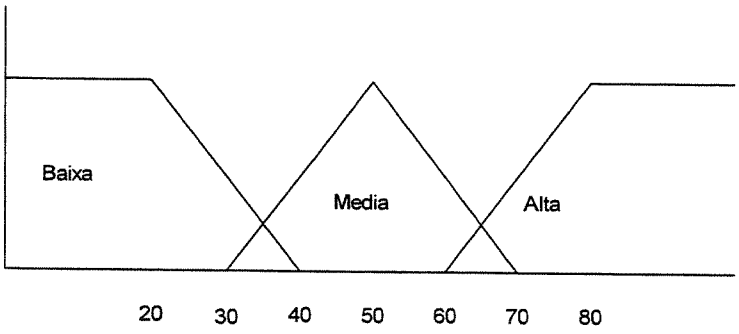


Figura 3.4 Conjunto nebuloso da prioridade

3.4.3. Prevenção de Bloqueio na Linha Ferroviária

Usualmente, escalona-se o deslocamento dos trens em uma linha ferroviária especificando preferências quando ocorrem encontros e ultrapassagens. O problema que se enfrenta é decidir se um trem pode continuar seu percurso ou deve ficar parado na sua locação atual. Parte desse problema consiste em verificar se é possível ou não deslocar um trem e, se houver deslocamento, não impossibilitar a movimentação dos outros trens, a menos que um deles se movimente no sentido contrário. Isto é conhecido como bloqueio de linha e é um problema encontrado em muitos modelos de simulação.

Petersen e Taylor [PET83] propõem condições para prever bloqueio de linha analisando o estado de toda a frota de trens ao longo da linha ferroviária.

Neste trabalho, propõe-se um algoritmo para prever bloqueio de linha. A idéia principal deste algoritmo é a de encontrar um *segmento livre* para um trem que se pretende deslocar – ao qual nós referenciaremos como *trem analisado* – e, em alguns casos, encontrar um *segmento livre* para o *trem contrário* (trem que está no sentido contrário) mais próximo dele.

Um *segmento livre* é definido como um segmento pelo qual um trem denominado *TREM* (*TREM* pode ser tanto o *trem analisado* ou o *trem contrário*) pode prosseguir sem interromper o deslocamento de trens viajando no sentido contrário. Um segmento é considerado como um *segmento livre* se satisfaz pelo menos uma das seguintes condições:

- é o segmento final da linha, levando em consideração o sentido de viagem do *TREM;*

- contém pelo menos um trecho livre e pelo menos um trecho ocupado por um trem viajando no mesmo sentido do *TREM;*
- contém mais de um trecho livre.

Para detalhar o algoritmo que encontre um *segmento livre* é necessário formalizar a representação dos seus elementos. A forma de representação escolhida é de um grafo. Um grafo **G(V,E)** é definido por um conjunto de vértices **V** e por um conjunto de arcos **E**. Em uma linha ferroviária os extremos dos segmentos são nós e os trechos dos segmentos são arcos. Os trechos que conectam dois segmentos duplos também são considerados arcos (*e.g* o arco f na Figura 3.6).

A Figura 3.5 apresenta uma linha ferroviária com quatro trens. A representação desta linha na forma de um grafo é apresentada na Figura 3.6. Para a linha da Figura 3.5 o grafo **G(V, E)** é caracterizado por 3.45 e 3.46 e a Tabela 3.1 abaixo.

$$\mathbf{V} = \{A, B, C, D, E, F\} \tag{3.45}$$

$$\mathbf{E} = \{a, b, c, d, e, f, g, h\} \tag{3.46}$$

onde:

Segmento 1	Segmento 2	Segmento 3	Segmento 4
a = (A,B)	c = (B,C)	d = (C,D)	g = (E,F)
b = (A,B)		e = (C,E)	h = (D,F)
		f = (D,E)	

Tabela 3.1. Arcos do grafo da Figura 3.6

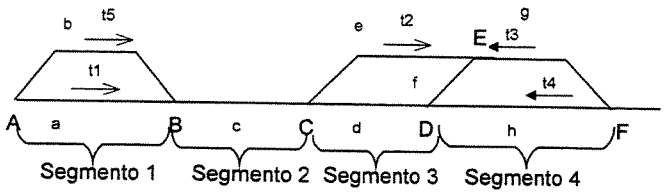


Figura 3.5 Exemplo de linha ferroviária

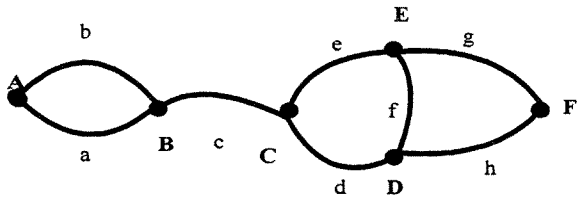


Figura 3.6 Representação da linha como um grafo

Um ou mais arcos adjacentes formam um caminho. Um caminho tem um sentido. O conjunto de caminhos entre nós **P(G),** contém os caminhos possíveis de um segmento até o próximo segmento. Estes caminhos são compostos por um arco ou por uma seqüência de arcos contendo pelo menos um elemento que conecta os segmentos com trechos paralelos múltiplos. Como sentido dos caminhos consideram-se como as mesmas direções dos trens analisados. Dado que o deslocamento de trens ocorrem em duas direções, os caminhos são definidos para ambas. Por exemplo, o conjunto de caminhos da Figura 3.6 está definido pela equação 3.47. Estes caminhos são caracterizados para as duas direções. Por exemplo, a Figura 3.7 e a Tabela 3.2 definem os caminhos no sentido *outbound* enquanto que a Figura 3.8 e a Tabela 3.3 os caracterizam no sentido *inbound.*

$$\mathbf{P(G)} = \{a1, a2, b1, c1, c2, c3, d1, d2, d3\} \tag{3.47}$$

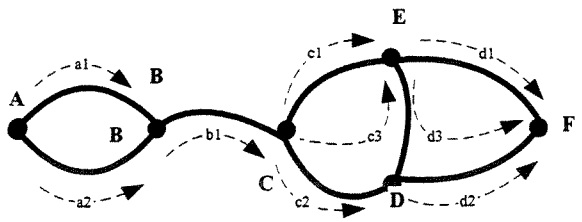


Figura 3.7 Caminhos da linha da Figura 3.5: sentido *outbound*

Segmento 1	Segmento 2	Segmento 3	Segmento 4
a1 =(A,B)	b1= (B,C)	c1 =(C,E)	d1 = (E,F)
a2 =(A,B)		c2 =(C,D)	d2 = (D,F)
		c3 =(C,E)	d3 = (E,F)

Tabela 3.2 Caminhos da Figura 3.7 e seus nós extremos

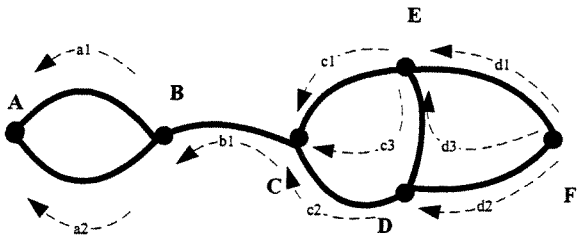


Figura 3.8 Caminhos da linha da Figura 3.5: sentido *inbound*

Segmento 1	Segmento 2	Segmento 3	Segmento 4
a1 =(B,A)	b1= (C,B)	c1 =(D,C)	d1 = (F,D)
a2 =(B,A)		c2 =(E,C)	d2 = (F,D)
		c2 =(E,C)	d2 = (F,D)

Tabela 3.3 Caminhos da Figura 3.8 e seus nós extremos

Para encontrar um *segmento livre* para um *TREM*, deve-se definir um peso w(i) para cada caminho $i \in$ **P(G)**. Um caminho terá um peso igual a:

- 1, se ele contém um trem viajando no mesmo sentido que *TREM*;
- -1, se ele contém um trem viajando no sentido oposto a *TREM*;
- 0, se ele está vazio.

Na busca por um segmento livre para o *trem analisado,* a designação de pesos deve ser realizada começando pelo caminho que este trem pretende ocupar, isto é, considera-se o *trem analisado* ocupando o segmento pretendido. No caso de se procurar por um *segmento livre* para o *trem contrário*, a designação de pesos se inicia no caminho por ele ocupado.

Por exemplo, considerando a Figura 3.5 e tomando como referência o deslocamento do trem t1 (*trem analisado*), alocado no caminho a1, para o caminho b1, os pesos dos caminhos são aqueles da Tabela 3.4. Para o *trem contrário* mais próximo ao *trem analisado,* o trem t4, os pesos dos caminhos são apresentados na Tabela 3.5.

Segmento 1	Segmento 2	Segmento 3	Segmento 4
w(a1) =0	w(b1)= 1	w(c1) = 1	w(d1) = -1
w(a2) =0		w(c2) = 0	w(d2) = -1
		w(c3) = 0	w(d3) = -1

Tabela 3.4 Pesos dos caminhos: o trem t1 é o *trem analisado*

Segmento 1	Segmento 2	Segmento 3	Segmento 4
w(a1) =0	w(b1)= -1	w(c1) = -1	w(d1) = 1
w(a2) =0		w(c2) = 0	w(d2) = 1
		w(c3) =0	w(d3) = 1

Tabela 3.5 Pesos dos caminhos: o trem t4 é o *trem contrário mais próximo*

Após representar a malha ferroviária por um grafo **G(V,E),** definir o conjunto de caminhos **P(G)** e seus pesos w(i), o algoritmo para encontrar um *segmento livre* para um trem *TREM* é o seguinte:

1. Seja *(Y', Z')* a posição inicial do trem *TREM*:

   $(Y', Z'); Y', Z' \in \mathbf{V}$

2. Construir o conjunto de caminhos a serem analisados:

   $CP = \{i \mid i = (Z', I), I \in \mathbf{V} \text{ and } i \in \mathbf{P(G)}\}$

3. Verificar o seguinte:

   3.1 Se $num(CP) = 0$ então um *segmento livre* foi encontrado; terminar.

   3.2. Se $[\exists\, i,j \mid w(i) = 1 \text{ e } w(j) = 0, i,j \in CP]$ então um *segmento livre* foi encontrado; terminar.

   3.3. Se $[\exists\, i,j \mid w(i) = 0 \text{ e } w(j) = 0, i,j \in CP\ ]$ então um *segmento livre* foi encontrado; terminar.

   3.4 Se $[\exists\, i \mid w(i) = 0 \text{ ou } w(i) = 1, i \in CP\ ]$ então:

   3.4.1 Seja Z tal que $i \in CP$ com $i = (J,Z)$ e $w(i) \in \{0,1\}$

3.4.2 Construir um novo conjunto de caminhos a serem analisados:

$$CP = \{i \mid i = (Z,I),\ i \in \mathbf{P(G)}\ and\ I \in \mathbf{V}\ \}$$

3.4.3. continuar (ir a 3.)

3.5. Caso contrário um *segmento livre* não foi encontrado; terminar

A função *num()* retorna a cardinalidade do conjunto dado como parâmetro. O passo 3 do algoritmo busca por um *segmento livre*. O passo 3.1 considera o caso de se encontrar o final da linha. O passo 3.2 verifica se foi encontrado um segmento com pelo menos um trecho livre e com um trecho ocupado por um trem viajando no mesmo sentido do trem para o qual se está aplicando o algoritmo. O passo 3.3 verifica se foi encontrado um segmento com mais de um trecho livre. O passo 3.4 verifica se existe ou um trecho com um trem viajando no mesmo sentido do trem para o qual se aplica o algoritmo, ou se existe um trecho livre. Se existir um trecho com estas características, um novo conjunto de caminhos *CP* é construído a partir do qual se procura um *segmento livre*. Se nenhuma destas condições é satisfeita, não existe um *segmento livre* (passo 3.5).

Como foi mencionado em parágrafos anteriores, as vezes é necessário encontrar um *segmento livre* para o *trem contrário* mais próximo ao *trem analisado*. Isto é necessário se, ao executar o algoritmo para encontrar um *segmento livre* para o *trem analisado*, este termina ao satisfazer a condição 3.2.

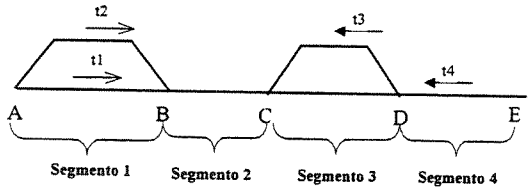


Figura 3.9 Linha ferroviária

A seguir se apresenta um exemplo da aplicação do algoritmo de prevenção de bloqueio considerando a linha e os trens da Figura 3.9. Esta linha contém 4 segmentos e 4 trens, 2 dos quais viajam no sentido *outbound* – t1 e t2 – e 2 no sentido *inbound* – t3 e t4. A Figura 3.10 mostra o grafo que representa esta linha e as equações 3.48 e 3.49 definem os nós e arcos deste grafo. O conjunto de caminhos **P(G)** está definido na equação 3.50

e na Tabela 3.6 e Tabela 3.7 define-se os caminhos para os sentidos *inbound* e *outbound* respectivamente.

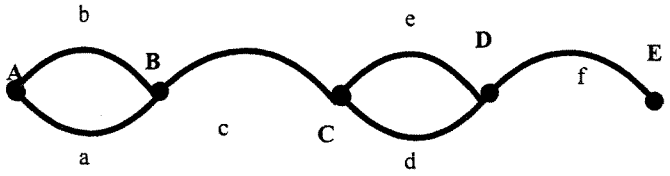


Figura 3.10 Grafo para a linha da Figura 3.9

$$\mathbf{V} = \{A, B, C, D, E\} \tag{3.48}$$

$$\mathbf{E} = \{a, b, c, d, e, f\} \tag{3.49}$$

$$\mathbf{P(G)} = \{a1, a2, b1, c1, c2, d1\} \tag{3.50}$$

Segmento 1	Segmento 2	Segmento 3	Segmento 4
a1 =(B,A)	b1= (C,B)	c1 =(D,C)	d1 = (E,D)
a2 =(B,A)		c2 =(D,C)	

Tabela 3.6 Nós extremos dos caminhos da Figura 3.10: sentido *inbound*

Segmento 1	Segmento 2	Segmento 3	Segmento 4
a1 =(A,B)	b1= (B,C)	c1 =(C,D)	d1 = (D,E)
a2 =(A,B)		c2 =(C,D)	

Tabela 3.7 Nós extremos dos caminhos da Figura 3.10: sentido *outbound*

Neste exemplo, considera-se o trem t4 como *trem analisado*. A Tabela 3.8 mostra os pesos dos caminhos de **P(G)** considerando o deslocamento do *trem analisado* enquanto que a Tabela 3.9 define os pesos dos caminhos considerando o *trem contrário* mais próximo (trem t1 ou t2). Observe que o considera-se o trem t4 ocupando o caminho c1.

Segmento 1	Segmento 2	Segmento 3	Segmento 4
a1 = -1	b1= 0	c1 = 1	
a2 = -1		c2 = 1	

Tabela 3.8 Pesos dos caminhos da Tabela 3.6: t4 *trem analisado*

Segmento 1	Segmento 2	Segmento 3	Segmento 4
a1 = 1	b1=0	c1 = -1	d1 =0
a2 = 1		c2 = -1	

Tabela 3.9 Pesos dos caminhos da Tabela 3.7: *trem contrário* t4

Executando o algoritmo e fazendo-se uso da Tabela 3.8, no passo 2 obtemos o conjunto CP = {c1, c2}. Este conjunto satisfaz a condição 3.4, logo um novo conjunto CP = {b1} é criado. Este conjunto, também satisfaz a condição 3.4, e obtém-se, novamente, um novo conjunto CP = {a1, a2}. Este conjunto não satisfaz nenhuma das condições impostas em 3.1, 3.2, 3.3 ou 3.4. Portanto, o algoritmo termina indicando que não é possível encontrar um *segmento livre* para o trem t4 e portanto não é aconselhável deslocá-lo pois pode ocasionar bloqueio da linha. Neste caso, já não é necessário aplicar o algoritmo para encontrar o *segmento livre* para o *trem contrário.*

Este algoritmo foi testado simulando a circulação de vários trens. Embora este algoritmo forneça bases suficientes para prever o bloqueio na linha ferroviária, ele é conservativo e apresenta algumas limitações no seu uso. Por isso, trens que poderiam ser movimentados acabam ficando parados por um tempo (exemplo trem t1 na Figura 3.5). Não obstante, a aplicação deste algoritmo tem um baixo custo computacional o que facilita o uso deste algoritmo em aplicações que por se mesmas já acarreiam um custo computacional considerável (e.g. um simulador).

3.5. Resumo

Neste capítulo apresentou-se o modelo de linha de Petersen e Taylor [PET82]. Este modelo foi descrito através dos seus elementos, suas respectivas variáveis e parâmetros e da sua lógica. A lógica é traduzida por expressões algébricas que relacionam as variáveis e

parâmetros dos componentes do modelo. Os componentes deste modelo são: linha ferroviária, trens despachados, trens imaginários, atividades de manutenção e tempos.

Tomando como base este modelo, definiu-se um modelo próprio. Este modelo acrescenta alguns itens ao modelo original. Os itens deste modelo são: malha ferroviária, trens, trens imaginários, atividades de manutenção, modelos de trens, modelos de percurso, modelos de escalas, perturbações em trens e tempos. A lógica deste modelo é similar à lógica do modelo original. A Tabela 3.10 mostra uma comparação dos elementos de ambos modelos.

Neste capítulo, apresentou-se também um algoritmo de prevenção de bloqueio da linha. O objetivo deste algoritmo foi encontrar um *segmento livre* para um trem a ser despachado e se é necessário para o trem contrário mais próximo.

No capítulo seguinte, apresenta-se a modelagem orientada a objetos do modelo proposto.

Modelo de Petersen e Taylor	**Modelo Proposto**
Linha Ferroviária	Malha Ferroviária
Conexões Físicas	Conexões Físicas
Trens	Modelos de Trens
Trens Imaginários	Modelos de Escalas
Atividades de Manutenção	Modelos de Percursos
Tempos	Trens Imaginários
	Tempos
	Atividades de Manutenção
	Perturbações dos trens

Tabela 3.10 Comparação entre os elementos do modelo original e o proposto

Capítulo 4

Modelagem Orientada a Objetos do Sistema

O modelo de linha proposto foi analisado e projetado empregando técnicas de orientação orientado a objetos, fazendo-se uso da *Unified Modeling Language* (UML) [UML97]. Na modelagem, seguiu-se como metodologia o Processo Evolutivo da Engenharia de Software [MEY97]. Os detalhes do desenvolvimento desta modelagem são fornecidos em [RON00b] e [RON00c].

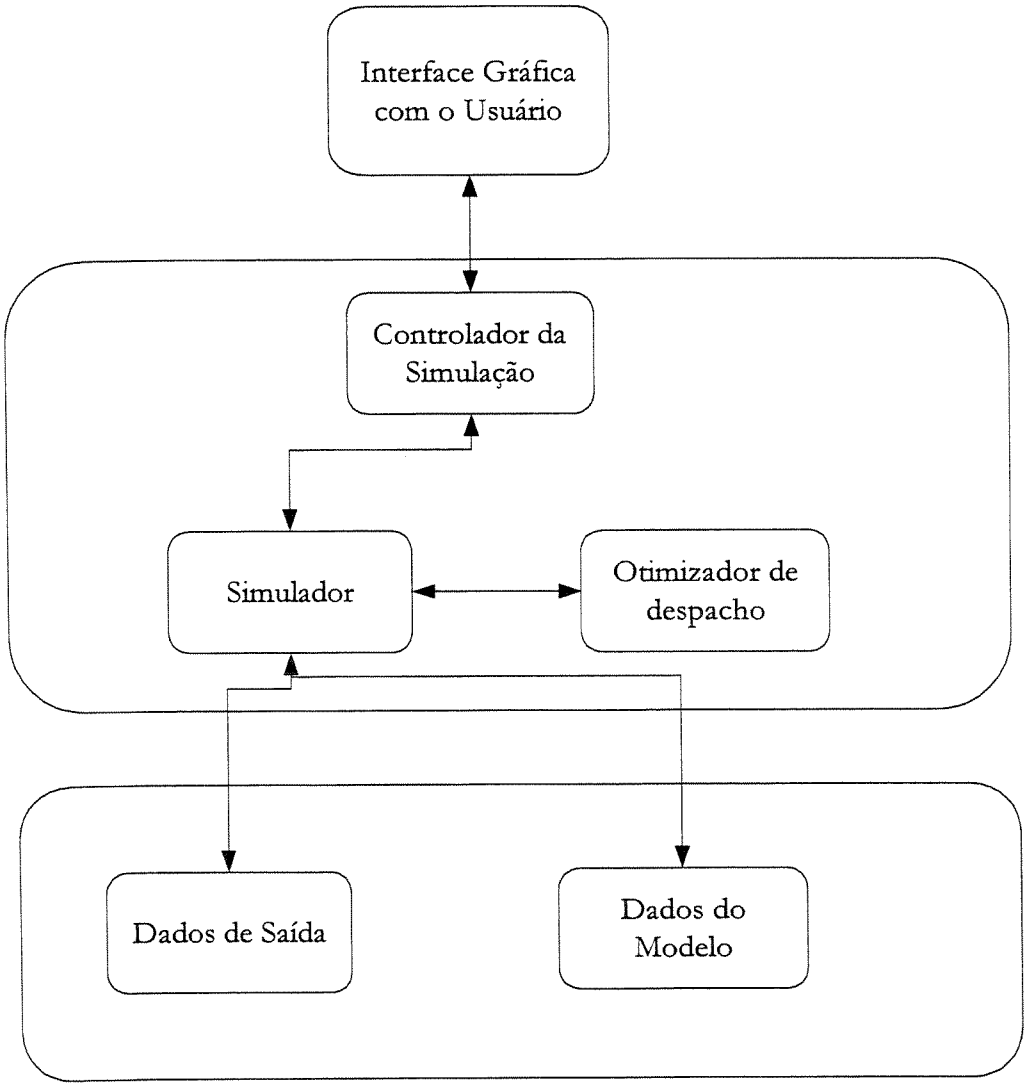


Figura 4.1. Esquema geral do sistema desenvolvido

Na Figura 4.1 apresenta-se a idéia geral do sistema desenvolvido. Esta figura mostra três níveis distintos. O nível superior representa a interface gráfica com o usuário. Os elementos envolvidos neste nível interagem diretamente com o usuário, permitindo-lhe atualizar dados do modelo de simulação, simular e observar os resultados. O segundo nível constitui os elementos envolvidos no processo de simulação. Neste nível observa-se o controlador da simulação, o processo de simulação (simulador propriamente dito) e o otimizador do despacho. O controlador é o encarregado de administrar a simulação, sendo o intermediário entre a interface gráfica e o processo da simulação. O processo da simulação executa a simulação, acessa os dados do modelo de linha, solicita ao otimizador os trens que terão locações reservadas e armazena os resultados (dados de saída). O otimizador seleciona um trem para reservar locação a partir de uma lista de trens concorrentes e de acordo com um objetivo a ser atingido. O terceiro nível contém os dados de entrada e de saída do sistema. Neste nível se encontram armazenados os dados da malha ferroviária, trens, modelos de trens, etc. Também encontram-se armazenados os dados de saída, e.g. resumo de eventos, atrasos sofridos pelos trens, etc.

4.1. Visões do Sistema

Um sistema é descrito sob um número de aspectos diferentes: funcional (estrutura estática e interações dinâmicas), não funcional (requisitos de tempo, requisitos de confiança, desenvolvimento, etc.) e aspectos organizacionais (organização do trabalho, mapeamentos de módulos de código, etc.). Cada visão representa uma projeção da descrição completa do sistema, mostrando um aspecto particular deste [ERI98].

Em [BOO98] definem-se cinco visões:

- Visão Caso de Uso (*use case*), é uma visão que mostra a funcionalidade do sistema como é percebida por usuários externos.
- Visão Lógica, é uma visão que mostra como uma funcionalidade é considerada pelo sistema, em termos de sua estrutura estática e de seu comportamento dinâmico.
- Visão Componente, é uma visão que mostra a organização dos componentes de código.
- Visão Concorrência, é uma visão que descreve as necessidades de concorrência e sincronização.

- Visão Desenvolvimento, é uma visão que descreve o mapeamento do software e hardware refletindo sua distribuição.

Cada visão é descrita através de diagramas que contém informações que enfatizam um aspecto particular do sistema. Os diagramas contém símbolos que representam elementos do modelo.

O objetivo deste capítulo e apresentar a modelagem orientada a objetos do modelo de linha proposto. O objetivo principal é mostrar as funcionalidades do modelo de linha na abordagem orientada a objetos. As visões que mostram estes aspectos do sistema são o *Caso de Uso* e a *Lógica*. Ambas são apresentadas neste capítulo.

4.2. Visão Caso de Uso

A visão Caso de Uso descreve a funcionalidade que o sistema deve apresentar, tal e como é percebido pelo ator externo, sem revelar sua estrutura interna. Um ator que interage com o sistema, pode ser tanto um usuário quanto um outro sistema. Esta visão é fundamental já que seu desenvolvimento orienta o desenvolvimento das outras visões. O objetivo final do sistema é prover a funcionalidade descrita nesta visão [ERI98]. A visão Caso de Uso é geralmente descrita usando diagramas de casos de uso.

Um diagrama de caso de uso é: um gráfico de atores, um conjunto de casos de uso incluído por um limite de domínio, participação e associações entre atores, assim como generalizações entre casos de uso [FUR98]. Cada caso de uso representa uma série de ações que os usuários podem executar ao interagir com o sistema. Estes podem ser descritos tanto textualmente quanto através de diagramas de atividade.

Existem diversas relações entre casos de uso [UML97]. Uma delas é a relação de *uso*, isto é, quando um conjunto de casos de uso tem um comportamento comum, este comportamento pode ser modelado em um caso de uso único que é utilizado por outros casos de uso [FUR97].

A seguir detalha-se os diagramas de casos de uso obtidos pela modelagem do sistema.

4.2.1. Caso de uso: Sistema

O diagrama de caso de uso do sistema, apresentado na Figura 4.2, foi construido extraindo as funcionalidades do sistema do modelo de linha proposto.

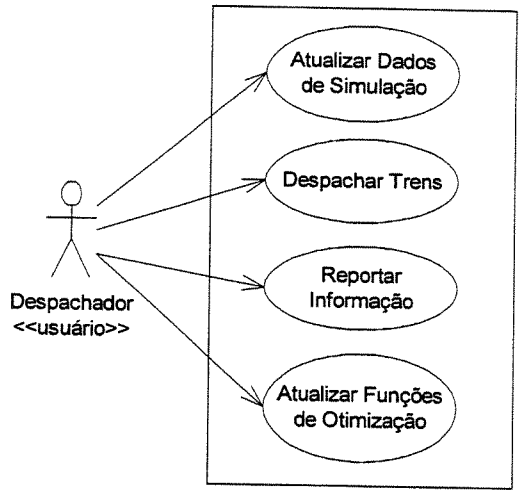


Figura 4.2 Diagrama de caso de uso: Sistema

Na Figura 4.1, observamos que o nosso sistema interage com o **Despachador**; que é o encarregado de atualizar a informação necessária para o processo de despacho de trens (**Atualizar Dados de Simulação**), de dar inicio à simulação (**Despachar Trens**), receber dados sobre os eventos ocorridos durante o despacho (**Reportar Informação**) e de atualizar funções de otimização de despacho (**Atualizar Funções de Otimização**). No diagrama, o **Despachador** age com um ator; **Atualizar Dados de Simulação, Despachar Trens, Reportar Informação** e **Atualizar Funções de Otimização** são os casos de uso que representam as funcionalidades do sistema. A seguir detalha-se cada um deles.

4.2.2. Caso de uso: Atualizar dados de simulação

O caso de uso **Atualizar dados de simulação** está relacionado com a definição dos dados a serem usados na simulação e com a facilidade de possibilitar ao usuário armazenar estes dados em um arquivo e lê-los. Os dados a serem usados na simulação referem-se aos elementos que compõem o modelo de linha proposto (malha ferroviária, trens, etc.).

O diagrama do caso de uso **Atualizar dados de simulação** é apresentado na Figura 4.3. Como podemos observar nesta figura, o caso de uso expande-se em 4 casos de uso: **Abrir arquivo de dados, Salvar arquivo de dados, Atualizar dados não dependentes** e **Atualizar dados dependentes**.

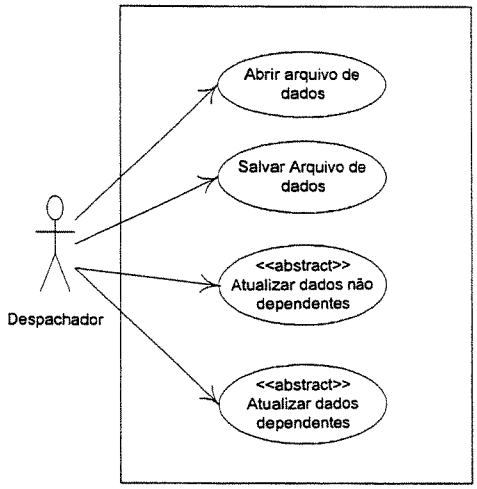


Figura 4.3 Diagrama de caso de uso: Atualizar dados de simulação

O caso de uso **Abrir arquivo de dados** tem como funcionalidade permitir ao usuário abrir um arquivo de dados e observar o seu conteúdo nas janelas de interface com o usuário criadas para tal objetivo.

O caso de uso **Salvar arquivo de dados** tem como funcionalidade permitir ao usuário salvar em um arquivo os dados da simulação que foram inseridos previamente nas janelas de interface.

O caso de uso **Atualizar dados não dependentes** é um caso de uso abstrato [ERI98]. Este caso de uso foi criado com a finalidade de modelar a funcionalidade de atualizar dados (inserir, modificar ou apagar) cuja definição seja independente de qualquer um outro dado. Por exemplo: para atualizar dados da malha ferroviária não é necessário conhecer previamente os dados dos modelos de trens. Este caso de uso mantém uma relação de *uso* com os casos de uso **Atualizar dados da malha ferroviária** e **Atualizar dados dos modelos de trens não dependentes** (Figura 4.4). Ambos casos de uso tem o mesmo comportamento. O primeiro deles tem como funcionalidade permitir a atualização de dados da malha ferroviária e o segundo, permitir a atualização de dados dos modelos de trens. O comportamento destes casos de uso é descrito usando o diagrama de atividades na Figura 4.5.

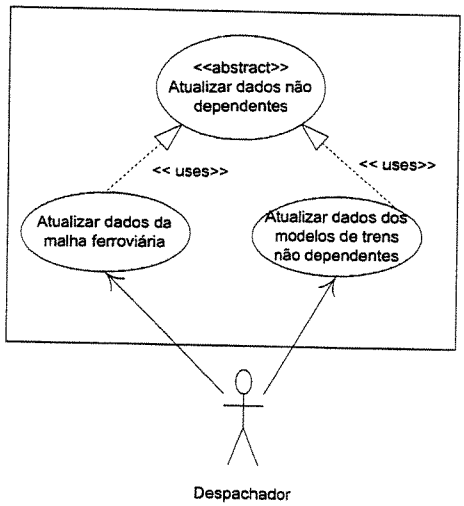


Figura 4.4 Diagrama de caso de uso: Atualizar dados não dependentes

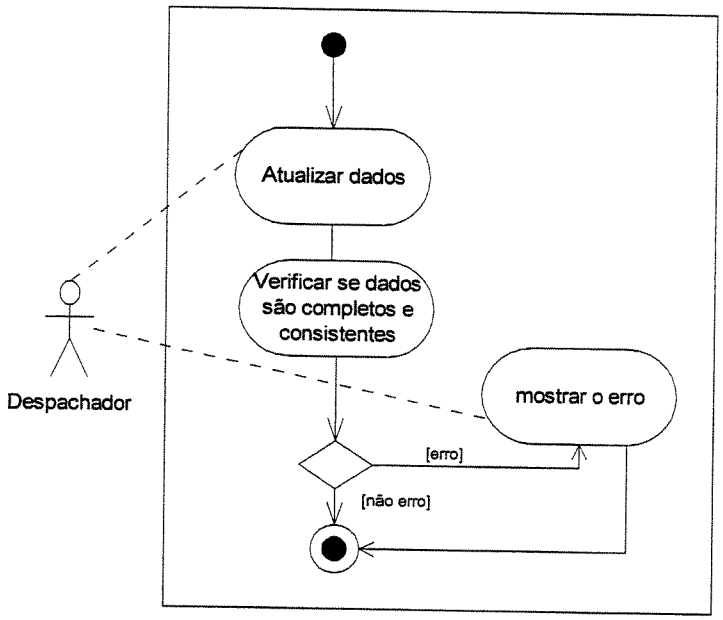


Figura 4.5 Diagrama de atividade: Atualizar dados não dependentes

O caso de uso **Atualizar dados dependentes** é também um caso de uso abstrato. Este caso de uso foi criado com a finalidade de modelar a funcionalidade de atualizar dados cuja definição depende de um outro dado. Por exemplo: para atualizar os dados das escalas que podem fazer os trens é necessário conhecer previamente os dados da malha ferroviária para garantir a consistência nos dados. Este caso de uso mantém uma relação de *uso* com os casos de uso: **Atualizar tempos médios de percurso,**

Atualizar dados de trens a despachar, Atualizar dados de perturbações que podem sofrer os trens, Atualizar dados de conexões físicas na malha ferroviária, Atualizar dados dos modelos de percurso, Atualizar dados dos modelos de escalas, Atualizar dados dos modelos de trens dependentes e **Atualizar atividades de manutenção** (Figura 4.6). O comportamento destes casos de uso é descrito usando um diagrama de atividades na Figura 4.7. Comparando este diagrama de atividade com o apresentado na Figura 4.5, observa-se que o primeiro inclui uma atividade mais do que o segundo. Esta atividade consiste em verificar a existência prévia de dados necessários.

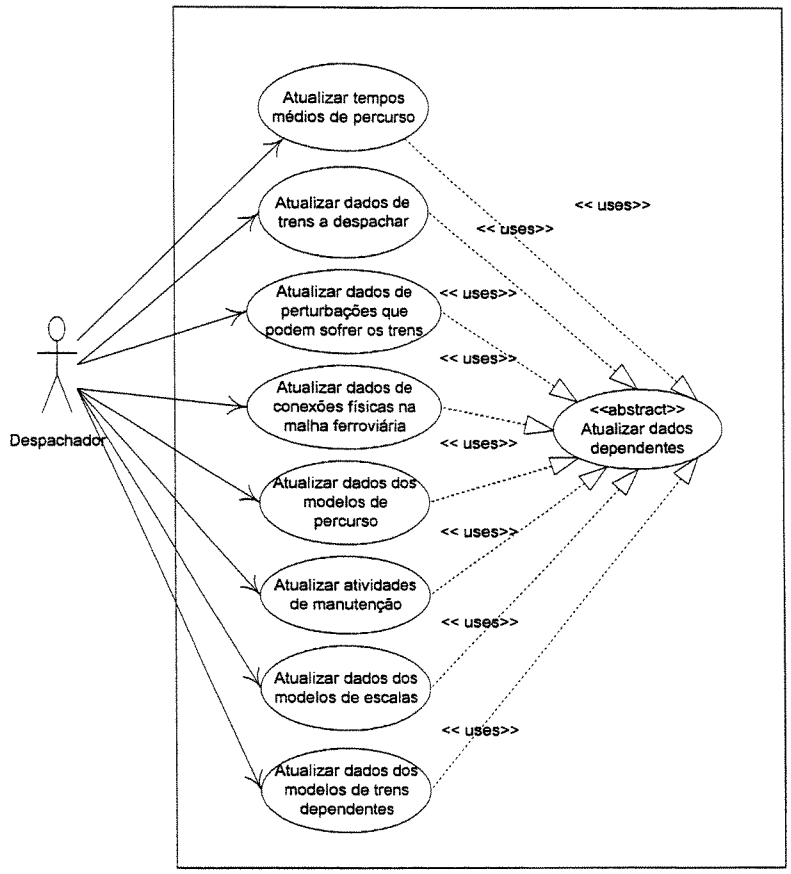


Figura 4.6 Diagrama de caso de uso: Atualizar dados dependentes

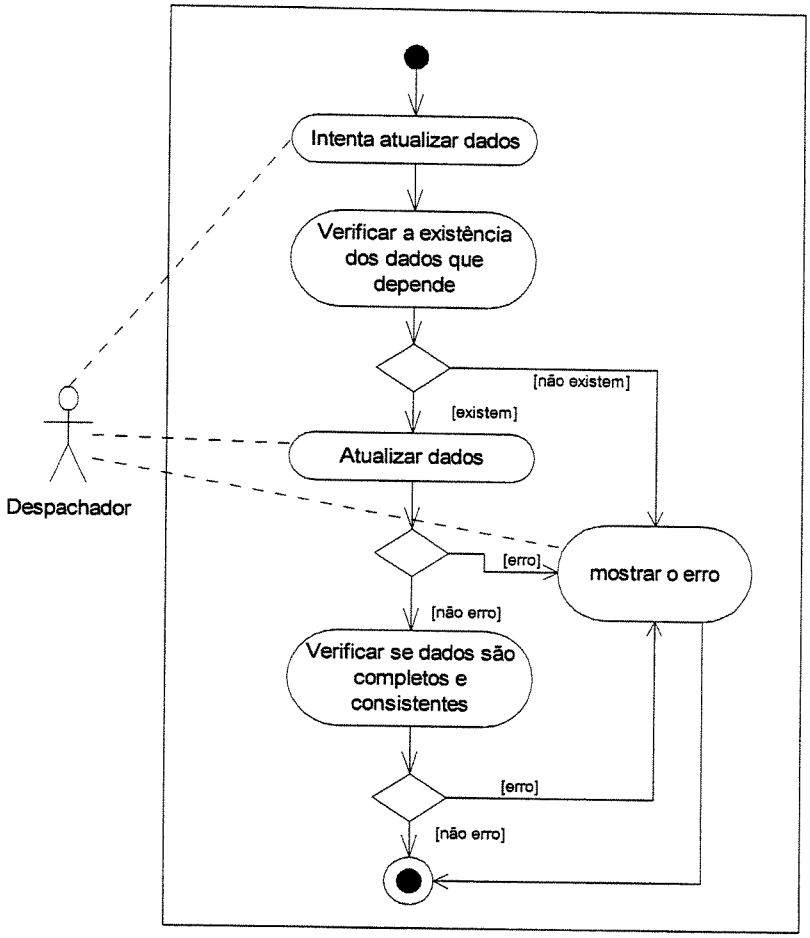


Figura 4.7. Diagrama de atividade: Atualizar dados dependentes

4.2.3. Caso de uso: Despachar trens

A funcionalidade deste caso de uso é permitir ao usuário configurar a(s) simulação(ões) de despacho de trens e de dar início a(s) mesma(s).

Configurar a simulação significa indicar: o horizonte de simulação (ex. número de dias a simular), quantas simulações se desejam fazer, qual arquivo contém os dados de entrada, que resultados são desejados, que método de otimização de despacho será usado e se são desejáveis simulações com dados determinísticos ou aleatórios.

O usuário envia um mensagem ao sistema para que este dê inicio às simulações. Esta funcionalidade engloba um grande grupo de processos internos, conforme descritos no capítulo anterior. A seguir, apresenta-se um resumo do caso de uso, assim como um

diagrama de atividade (Figura 4.8) com o objetivo de mostrar a seqüência de ações internas.

Sendo o sistema modelado um sistema a eventos discretos, o relógio do sistema avança de acordo com a ocorrência de eventos, eventos estes que correspondem ao término da permanência de um trem em uma locação, ao início do percurso de um trem ou a uma perturbação que um trem pode sofrer.

Se o evento refere-se a uma perturbação no trem, esta perturbação é refletida no estado do trem. Caso que o evento esteja relacionado com um trem imaginário este é descartado e o tempo avança para o evento seguinte.

Caso o evento esteja relacionado com um trem que completou sua viagem inicial modifica-se seu estado para refletir sua nova condição e avança-se o tempo até o evento seguinte. Caso contrário, a rotina de despacho decidirá se o trem avança até um trecho específico no segmento seguinte do seu percurso ou se o trem é atrasado na posição atual.

A rotina de despacho começa determinando o conjunto de trens que estão competindo por usar um mesmo segmento no mesmo intervalo de tempo. Todos os membros deste conjunto tem direito ao uso do próximo segmento; não obstante, alguns destes trens são "melhores" que outros. Para estes "melhores" trens reserva-se locações. Logo, determina-se as locações disponíveis e seleciona-se uma delas. Se o deslocamento do trem atual até a locação selecionada provoca bloqueio de linha, então escolhe-se outra locação.

O estado do trem é atualizado para refletir o novo estado e o tempo no qual uma nova atividade será completada. Além disso, se o trem ocupa uma nova locação, um trem imaginário é criado para ocupar a locação anterior até que esteja disponível para ser usado por outro trem. A seguir o tempo avança até o próximo evento.

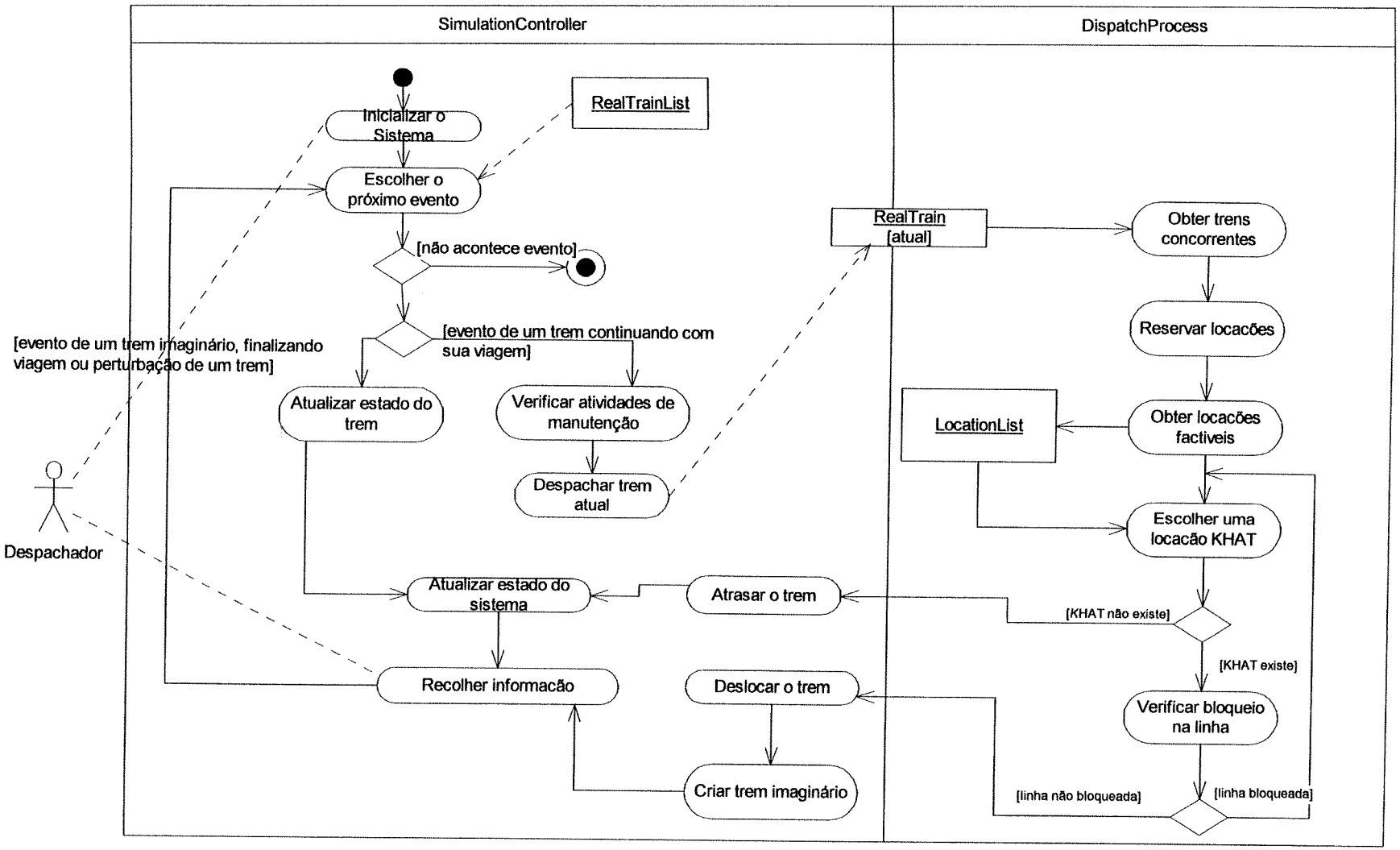


Figura 4.8 Diagrama de atividade: Despacho de trens

4.2.4. Caso de uso: Reportar informação

Este caso de uso tem como funcionalidade permitir ao usuário observar os diferentes resultados obtidos durante a simulação e armazenar estes resultados. No capítulo seguinte são apresentados exemplos de resultados fornecidos pelo sistema implementado.

4.2.5. Caso de uso: Atualizar funções de otimização

Este caso de uso tem como funcionalidade permitir ao usuário atualizar – abrir e salvar arquivos, modificar ou criar novas funções – funções de otimização a serem usadas no despacho de trens. Estas funções são escritas na linguagem especificada no Apêndice e são armazenadas em arquivos de extensão "opt" para serem usadas durante a simulação de despacho de trens.

Os quatro últimos casos de uso apresentados representam o comportamento do sistema tal como o usuário o deseja. O sistema computacional resultante pode ser avaliado conferindo o fornecimento deste comportamento. A seguir apresenta-se a visão lógica deste sistema.

4.3. Visão Lógica do Sistema

A visão lógica descreve como a funcionalidade do sistema é fornecida. Esta visão transparece o interior do sistema e descreve tanto a estrutura estática (classes, objetos e relações) quanto colaborações dinâmicas que ocorrem quando os objetos enviam mensagens entre eles para fornecer uma função [ERI98].

A estrutura estática descreve entidades como elementos discretos de modelagem, mas não contém detalhes do seu comportamento dinâmico [RUM99]. Esta estrutura é descrita utilizando diagramas de classes e objetos.

O diagrama de classes mostra a estrutura estática das classes do sistema. Estas podem estar relacionadas umas com as outras de diversas formas: associação, dependência e pacote [ERI98]. O diagrama de objetos é uma variação deste diagrama que pode apresentar um possível retrato da execução do sistema em um determinado momento.

O comportamento dinâmico do sistema é descrito em UML usando diagramas de estado, atividade, seqüência e colaboração. Estes diagramas mostram como os objetos interagem dinamicamente em diferentes momentos durante a execução do sistema.

Para descrever o comportamento dinâmico do sistema não é necessário utilizar todos os diagramas que o podem descrever. A escolha do diagrama ou diagramas a serem usados depende do aspecto que se deseja focalizar.

O diagrama de estados é o encarregado de definir todos os estados possíveis de um sistema; isto é problema para um sistema de grande porte pois o crescimento exponencial do número de estados pode tornar a análise complexa e difícil[FUR97].

O diagrama de atividades é uma forma especial de diagrama de estados que permite modelar cálculos e fluxos de dados [RUM99].

Diagramas de seqüência e diagramas de colaboração expressam informação semelhante mas a apresenta de modo diferente. Para decidir qual diagrama deve ser usado para estudar uma interação, como regra geral pode-se escolher o diagrama de colaboração quando o objeto e seus vínculos facilitam a compreensão da interação, e escolher o diagrama de seqüência somente se a seqüência necessita ser evidenciada [FUR97].

Para descrever o comportamento do sistema optou-se pelo diagrama de colaborações, pois este diagrama enfatiza o contexto do problema e mostra as interações entre as classes definidas.

4.3.1. Estrutura Estática do Sistema

Nesta seção apresenta-se a estrutura estática do sistema implementado. Conforme mencionado acima, a estrutura estática do sistema é descrita através de diagramas de classes ou de objetos.

As classes do sistema implementado estão divididas em cinco tipos:

- Classes de interface com o usuário: são aquelas que possibilitam a construção da interface gráfica com o usuário;
- Classes de fonte de dados: são aquelas que contém a informação relacionada com o modelo de linha proposto;
- Classes do domínio do problema: são aquelas que estão relacionadas com o problema de despachar trens em uma malha ferroviária;

- Classes de resultados: são aquelas que armazenam os resultados obtidos durante a(s) simulação(ões);
- Classes de apoio: são aquelas classes que estão relacionadas com a ferramenta desenvolvida para interpretar arquivos de texto que contém funções de otimização de despacho.

Estes cinco tipos de classes foram agrupadas em *packages* ou subsistemas. Os *packages* são mecanismos de propósito geral para organizar elementos dentro de grupos semanticamente relacionados[ERI98]. A Figura 4.9 mostra os *packages* do sistema assim como a classe **SystemController** que exerce controle sobre todos eles. O *package* **Interpreter** agrupa as classes de apoio, **Data** agrupa as classes fonte de dados, **Domain** agrupa as classes do domínio do problema, **ReportInformation** agrupa as classes de resultados e **GUI** agrupa as classes de interface com o usuário.

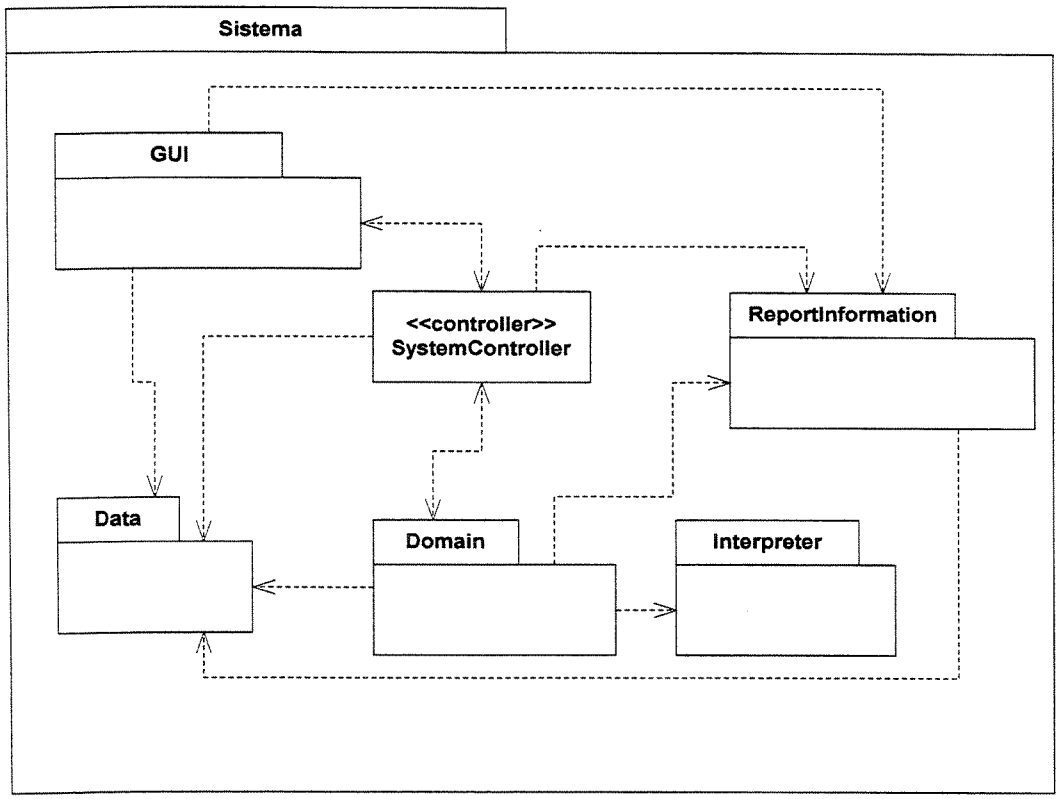


Figura 4.9 *Package* Sistema

O *package* **GUI,** por sua vez, está dividido em outros 5 *packages* (Figura 4.10). O *package* **Report** agrupa as classes de interface relacionadas com a apresentação dos

resultados ao usuário, **Insert** agrupa as classes de interface relacionadas com a atualização de dados do modelo de linha pelo usuário, **Simulation** agrupa as classes de interface relacionadas com a configuração de parâmetros da simulação e com o início da simulação, **Optimization** agrupa as classes de interface relacionadas com a definição de funções de otimização pelo usuário e, finalmente, **Basics** agrupa as classes de interface que servem de apoio para outras interfaces.

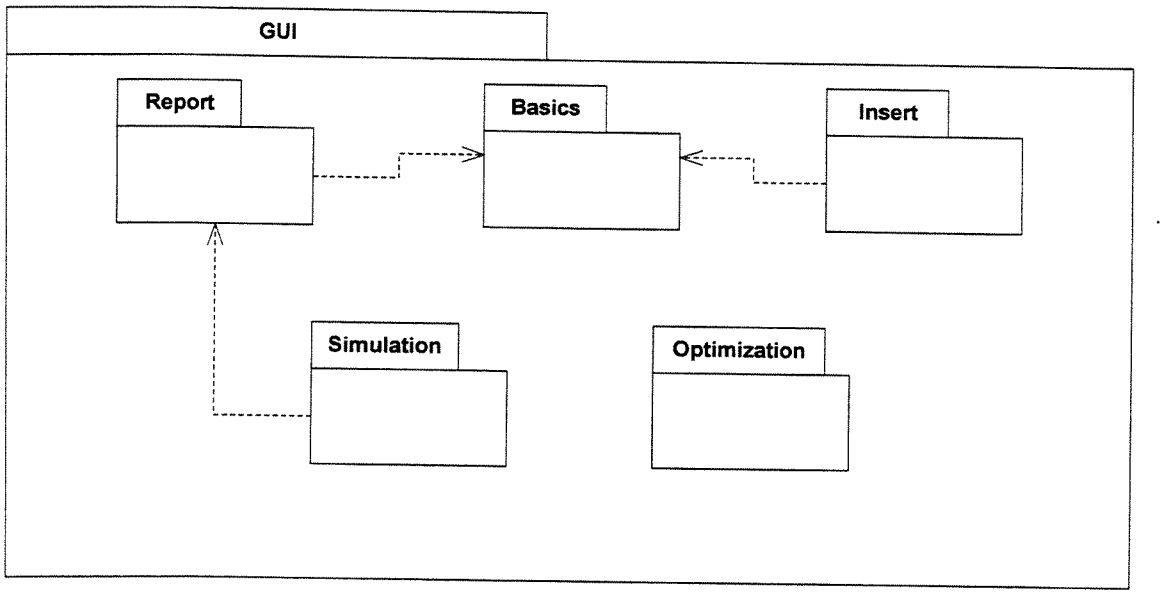


Figura 4.10 *Package* GUI

Neste capítulo estaremos voltados para as classes fonte de dados e do domínio do problema. Estas classes são as que detalham tanto os componentes do modelo de linha proposto quanto a lógica que os relaciona. Em [RON00c] apresenta-se uma descrição detalhada das classes de interface de usuário e de resultados.

Os diagramas de classes aqui apresentados mostram somente o nome das classes e suas relações, mas não apresentam nem seus atributos nem seus métodos. Uma descrição detalhada destas classes encontra-se em [RON00b] e [RON00c]. Algumas classes aparecem mais de uma vez nos diagramas de classe por razões de clareza na apresentação das associações.

4.3.1.1. Classes da Estrutura do Modelo de Linha Proposto

Nesta seção apresenta-se as classes que modelam os componentes do modelo de linha proposto. A Figura 4.11 mostra seu diagrama de classes. A seguir descreve-se cada uma das classes do diagrama e suas associações.

A classe **DataController** é uma classe criada para controlar as classes que modelam os componentes do modelo de linha.

A classe **RailLine** é uma classe que modela ou a malha ou a linha ferroviária. Uma linha ou malha ferroviária é modelada como uma composição de segmentos (**Segment**) interligados.

Um segmento é modelado pela classe **Segment**. Um segmento é composto por uma lista de locações **LocationList**. Uma locação é modelada pela classe **Location** que está relacionada com um segmento.

ConexionList é uma lista de conexões (**Conexion**), no sentido *inbound* ou no sentido *outbound* entre locações. A classe **Conexion** representa a conexão física possível entre duas locações.

A classe **Location** mantém duas associações com a classe **ConexionList**. Uma das associações modela as conexões físicas da locação no sentido *inbound* e a outra no sentido *outbound*.

Uma série de atividades de manutenção (**ManteinanceList**) podem ser realizadas durante o despacho de trens. Cada uma destas atividades (**Manteinance**) tem associada uma locação onde a manutenção será realizada.

O modelo de linha proposto contempla modelos de percurso. Este componente é modelado pela classe **JourneyPattern**. Um modelo de percurso é uma agregação ordenada de segmentos.

O modelo de linha contempla também a existência de modelos de escalas. Este componente é modelado pela classe **StopPattern**. Esta classe contém uma lista de escalas. Uma escala é modelada pela classe **Station**. Esta classe está associada com a classe **Segment**, o segmento onde será realizada a escala.

Também existem modelos de trens incluídos na classe **TrainPattern**. Esta classe pode manter uma associação com ela mesma. Se um modelo de trem A está associado a um modelo de trem B, isto significa que os tempos de percurso do modelo de trem A serão determinados a partir dos tempos de percurso do modelo de trem B.

A classe **LocationTime** é o resultado de uma associação binária entre a classe **Location** e a classe **TrainPattern**, e representa o tempo que um modelo de trem leva para atravessar uma locação.

A relação de trens despachados na malha é modelada pela classe **RealTrainList**. Cada um destes trens é modelado pela classe **RealTrain**. Esta classe tem duas associações com a classe **TrainPattern,** duas outras com a classe **StopPattern**, e representa os modelos de trens e escalas a seguir quando viaja no sentido *inbound* e *outbound.* Este trem seguirá um trajeto e, portanto, tem associado a ele a classe **JourneyPattern**. A classe **RealTrain** pode manter uma associação com ela mesma. Se esta associação é mantida, o horário de partida do trem depende do horário de partida de outro trem.

Durante o despacho, os trens podem sofrer perturbações. A relação de perturbações de trem que podem ocorrer é modelada pela classe **TrainDisruptionList**, sendo cada perturbação modelada pela classe **TrainDisruption**. Esta última classe tem uma associação com um trem despachado.

A classe **Config** contém informação relacionada com os parâmetros da simulação.

Algumas das classes mostradas no diagrama implementam a interface **Comparable**. Esta interface permite que objetos sejam comparados uns com outros e possibilita a ordenação de uma lista de elementos. Por exemplo, uma lista de objetos da classe **RealTrain** podem ser comparados e ordenados considerando-se o horário de partida.

Todas as classes apresentadas na Figura 4.11 estão inclusas dentro do *package* **Data.**

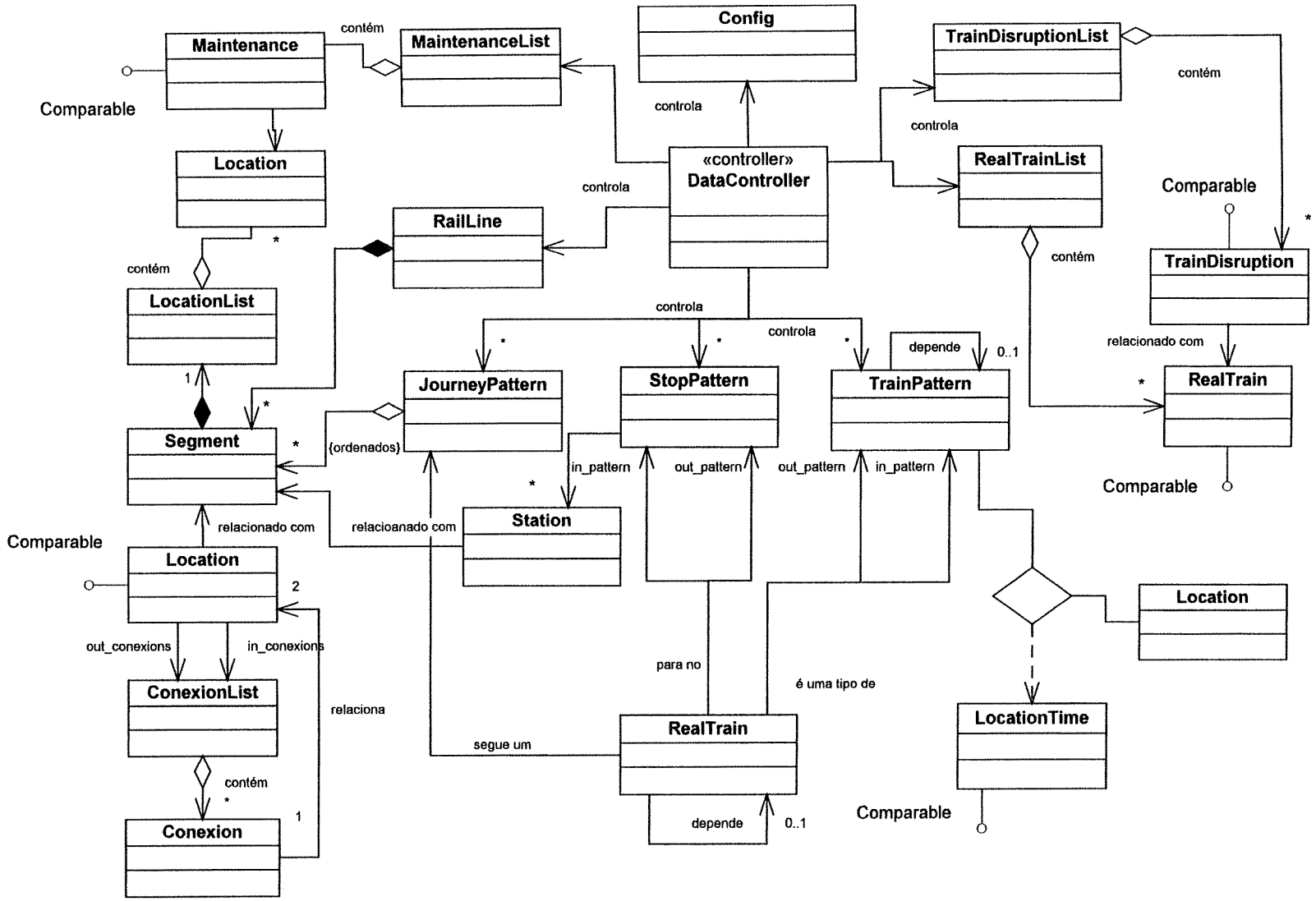


Figura 4.11 Diagrama de classes: Elementos do modelo de linha proposto

4.3.1.2. Classes da Lógica do Modelo de Linha Proposto

Esta seção apresenta as classes que modelam a lógica do modelo de linha proposto. A Figura 4.12 mostra seu diagrama de classes, onde aparecem algumas das classes do *package* **Data**. A seguir explicita-se cada uma das classes do diagrama e suas associações.

A classe que controla o despacho de trens é **SimulationController.** Esta classe tem acceso tanto ao controlador de dados de entrada (**DataController**), quanto ao controlador de dados de saída (**ReportDataController**). Ela interage com a classe **DataController** para obter informação relacionada com os dados da simulação de despacho de trens (número de dias simulados, malha ferroviária, etc.). Interage também com a classe **ReportDataController** para comunicar os eventos ocorridos durante a(s) simulação(ões).

A lógica do modelo de linha proposto contempla a existência de dois tipos de trens: trens despachados na linha ferroviária e trens imaginários. Ambos tipos de trens compartilham características comuns. Ambos estão ocupando alguma locação, possuem um intervalo de tempo no qual completarão suas próximas atividades e podem estar parados ou não. Estas características comuns deram lugar a criação da uma superclasse **Train**, da qual herdam atributos e métodos as classes **RealTrain** (que modela um trem despachado na malha) e **PhantomTrain** (que modela um trem imaginário). **TrainList** é uma agregação de objetos da classe **Train** ou especializações dela.

O estado da malha ferroviária pode ser obtido através da classe **TrainsInSegment**. Esta classe é uma associação binária entre cada segmento da malha ferroviária e a lista de trens à eles alocados.

Retomando a classe **SimulationController** observamos que ela mantém uma associação com a classe **TrainList**, representando os trens que estão na malha ferroviária, tanto os que foram despachados quanto os imaginários. A classe **SimulationController** gerencia o despacho de um trem atual **RealTrain**. Esta classe conhece o segmento onde está o trem localizado e os dois segmentos seguintes de seu percurso. Quando a classe **SimulationController** necessita obter uma locação para deslocar o trem atual, ela cria uma instância da classe **DispatchProcess**, a classe que modela a rotina de despacho.

A classe **DispatchProcess** tem uma associação com a classe **RealTrain** e com **Location**, pois ela é a encarregada pela busca por locação para ser ocupada por um trem. Na

busca por esta locação ela cria três instâncias da classe **LocationList** representando as locações que são disponíveis, as que ocasionam bloqueio na malha ferroviária e as que são reservadas para serem usadas por outros trens. Nesta busca, cria-se também a lista de trens – instâncias da classe **RealTrainList** – que estão concorrendo pelo uso do segmento para o qual será despachado o trem atual, e outra lista de trens que tem locações reservadas. Note-se que somente instâncias da classe **RealTrain** são capazes de reservar locações ou de concorrer pelo uso de um segmento. Por isto a lista de trens que reservam locações e a lista de trens concorrentes são instâncias da classe **RealTrainList** e não da classe **TrainList**. A classe **DispatchProcess** está também associada com a classe **SimulationController** para ter acesso ao estado da malha ferroviária e à lista de trens alocados.

4.3.2. Comportamento Dinâmico do Sistema

A dinâmica de um sistema é estabelecida pela forma em que os objetos se comunicam e os efeitos desta comunicação. Na modelagem orientada a objetos, a comunicação entre objetos é realizada através de mensagens enviadas de um objeto a outro. Essas mensagens, evidenciam as operações que são requisitadas a cada objeto, e portanto, as operações das respectivas classes.

Como já foi mencionado, a dinâmica do sistema será apresentada fazendo uso de diagramas de colaboração. O objetivo é apresentar as colaborações entre classes que simulam o despacho de trens na malha ferroviária. A partir destes diagramas pode-se identificar as operações que determinam o comportamento das classes.

4.3.2.1. Colaborações entre classes: Início da simulação

O diagrama de colaborações apresentado na Figura 4.13 mostra as mensagens entre classes que ocorrem quando o usuário dá início ao processo de simulação. Este diagrama inclui uma instância da classe (objeto) **SimulationConfigPanel,** inclusa no *package* **GUI.Simulation**, que implementa a janela que o usuário visualiza quando quer iniciar a(s) simulação(ões). Inclui também uma instância da classe **MainFrame** que modela o controlador das classes de interface com o usuário. Esta classe tem acesso ao controlador do sistema computacional modelado pela classe **SystemController**.

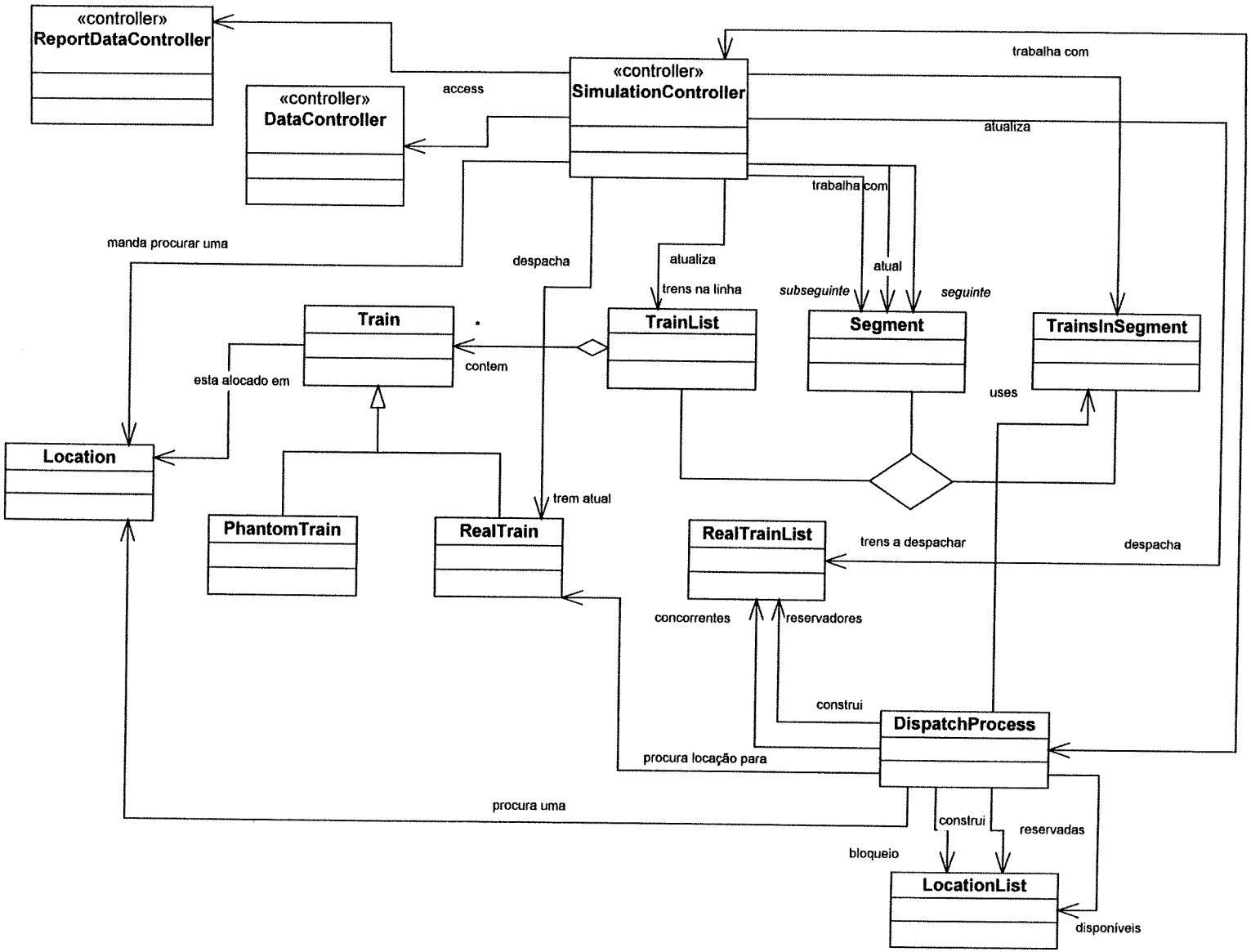


Figura 4.12 Diagrama de classes: Lógica do modelo de linha proposto

Esta atividade inicia com uma mensagem **PushSimulationButton()** enviada pelo usuário dando inicio à(s) simulação(ões). Esta mensagem é capturada pela instância da classe **:SimulationConfigPanel**, quem é a encarregada de enviar a mensagem **initProcess(fileName:String)** ao controlador da interface **:MainFrame** quem, por sua vez, envia a mesma mensagem ao controlador do sistema **:SystemController** para iniciar a(s) simulação(ões).

O controlador do sistema cria o controlador de dados de entrada **DATA**: **DataControler** e solicita a este – usando a mensagem **buildData(fileName:String)** – transformar o conteúdo de um arquivo de dados nos objetos correspondentes. Percorrendo o conteúdo do arquivo de dados, o controlador de dados de entrada cria a malha ferroviária (**createRailLine**()), os modelos de trens (**createTrainModels**()), os modelos de percursos, (**createJourneys**()), os modelos de escalas (**createStopModels**()), os tempos de percurso dos modelos de trens (**createTrainsTimes**()), os trens a serem despachados (**createTrainsToDispatch**()), as atividades de manutenção (**createMaintenanceActivities**()) assim como as perturbações que podem sofrer os trens (**createTrainDisruptions**()).

Após esta mensagem, o controlador do sistema cria o controlador de relatórios **REPORT:ReportDataController** e o controlador da simulação **Controller:SimulationController**. A seguir, o controlador do sistema envia a mensagem **initProcess()** ao controlador da simulação para o início da(s) simulação(ões). Esta última mensagem será analisada na seção seguinte.

4.3.2.2. Colaborações entre classes: Simulação

Nesta seção mostra-se como as classes colaboram com o controlador da simulação **Controller:SimulationController** para fornecer resposta à mensagem **initProcess()** enviada pelo controlador do sistema **:SystemController.** O diagrama de colaboração é apresentado na Figura 4.14.

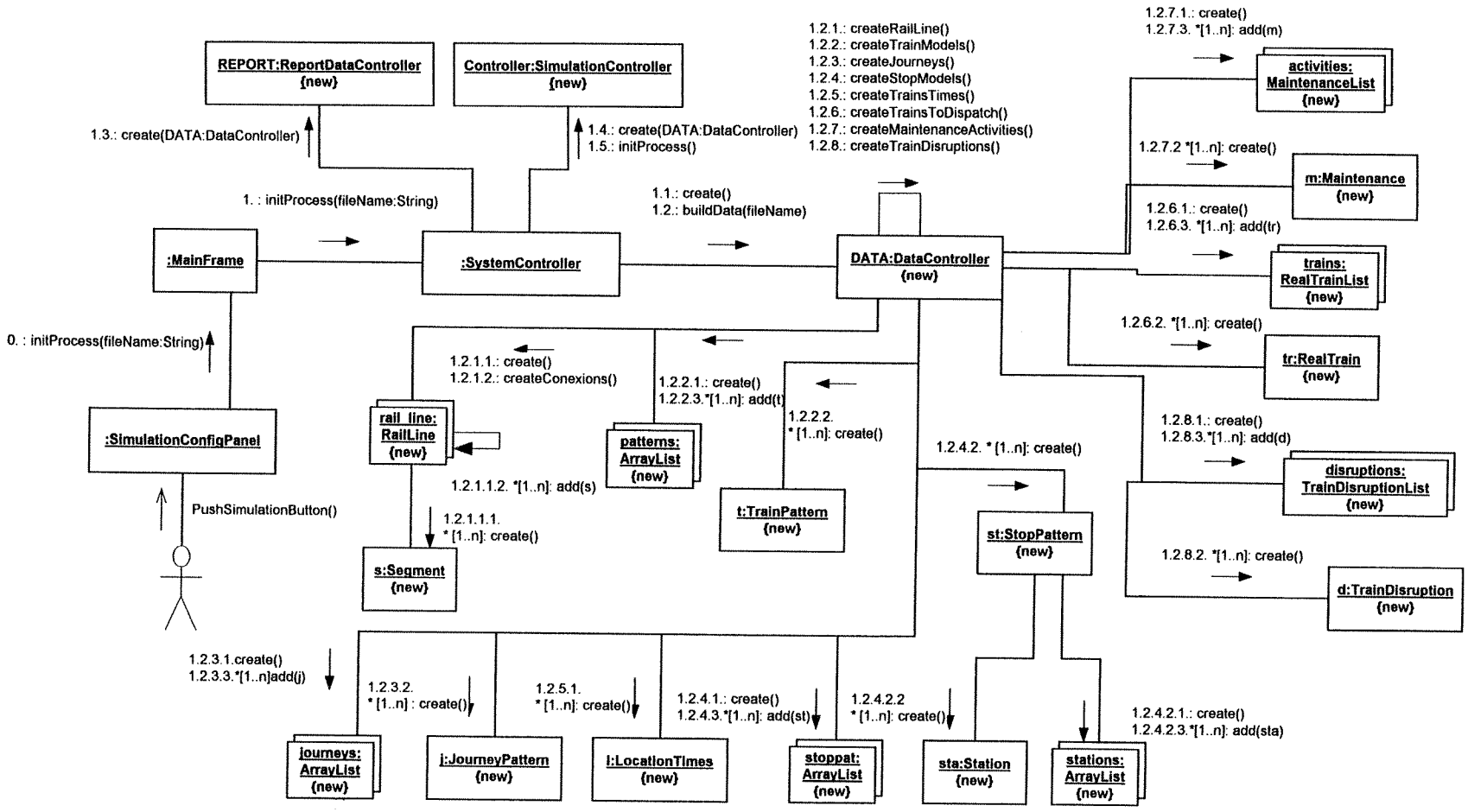


Figura 4.13 Diagrama de colaboração: Início da simulação

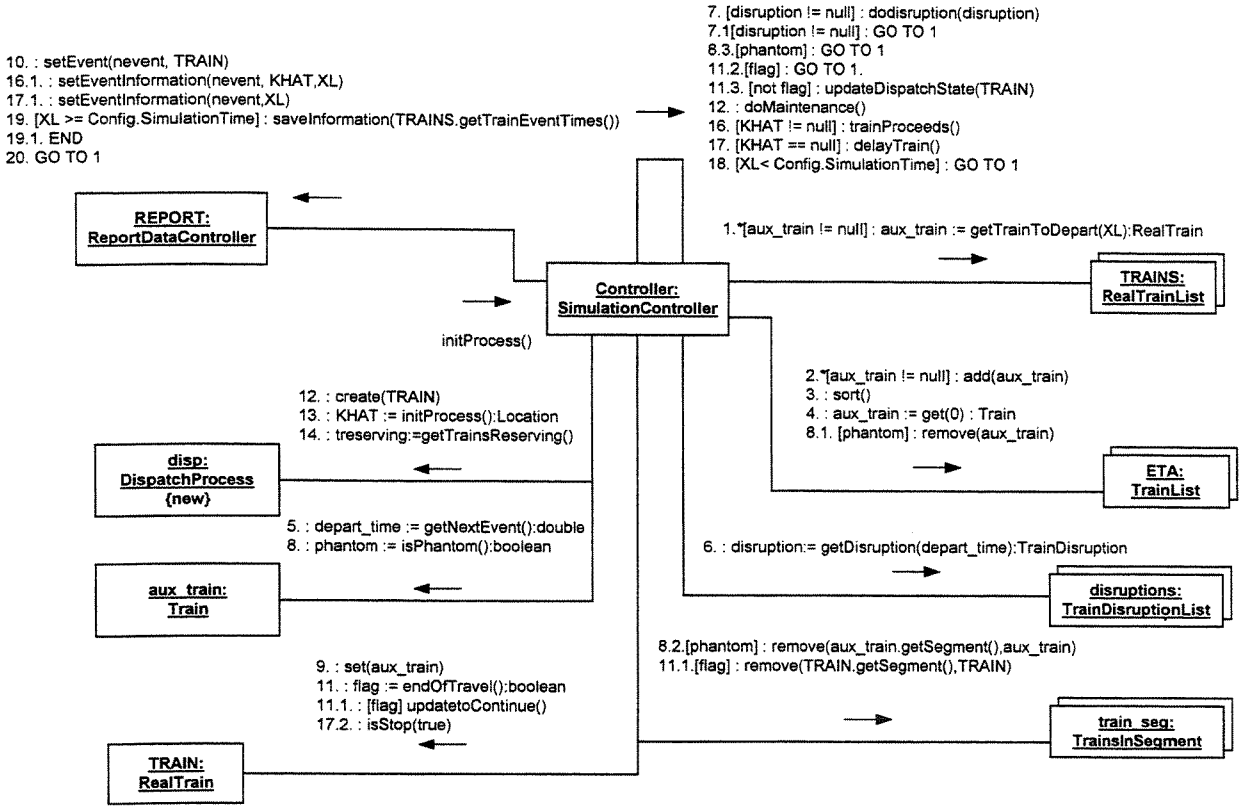


Figura 4.14 Diagrama de colaboração: Simulação

O controlador da simulação envia à lista de trens a serem despachados – **TRAINS:RealTrainList** – a mensagem **aux_train:= getTrainToDepart(XL):RealTrain** para que este devolva um próximo trem a partir. O parâmetro **XL** desta mensagem representa o momento no qual acontecerá o próximo evento do sistema. O trem **aux_train** é inserido na lista de trens na malha ferroviária **ETA:TrainList**. Esta mensagem é repetida até que não existam mais trens a partir.

A seguir o controlador da simulação envia a mensagem **sort()** à lista que contém os trens na malha ferroviária **ETA:TrainList**, para estes sejam ordenados considerando-se a hora na qual a próxima atividade terminará. Após a ordenação da lista, o controlador solicita à lista o trem que ocasionará o próximo evento (**aux_train**) e obtém a hora na qual este trem terminará a sua atividade atual usando a mensagem **depart_time:= getTNextEvent():double**.

Prosseguindo, o controlador da simulação envia a mensagem **disruption:= getDisruption(depart_time):TrainDisruption** à lista de perturbações de trens – **disruptions:TrainDisruptionList** – para verificar se algum trem sofrerá uma perturbação antes que este evento ocorra. Se um trem tem que ser perturbado a perturbação é realizada e o sistema procura o próximo evento. As mensagens que sucedem entre as classes quando um trem é perturbado são apresentadas na seção 4.3.2.3.

Se não houver perturbação, o controlador analisa se **aux_train:Train** é um trem imaginário usando a mensagem **isPhantom()**. Se o trem for imaginário, ele é removido tanto da lista de trens na linha ferroviária **ETA:TrainList**, quanto do estado da malha ferroviária **train_seg:TrainsInSegment** e o sistema trata o próximo evento.

Quando o trem **aux_train** for um trem despachado, o controlador da simulação designa valor a **TRAIN:RealTrain**, o qual representa o próximo trem a ser despachado. O controlador da simulação verifica se este trem chegou ao segmento final de seu percurso usando a mensagem **flag := endOfTravel():boolean**. Se a resposta for positiva, o trem é removido do estado da malha ferroviária, o seu estado é modificado usando a mensagem **updatetoContinue()**, e o controlador da simulação procura pelo próximo evento. A modificação do seu estado é baseada na mudança do seu sentido de viagem e nas operações necessárias para armazenar informação relacionada com sua viagem.

Se **TRAIN** não tiver chegado ao final de seu percurso, o estado da simulação é atualizado (**updateDispatchState(TRAIN)**) modificando-se o valor de **XL**.

Em seguida, o controlador envia a auto-mensagem **doMaintenance**() para verificar se serão realizadas atividades de manutenção. As colaborações entre objetos que ocasionam esta auto-mensagem são discutidas na seção 4.3.2.4

A atividade seguinte do controlador da simulação é criar uma instância da classe **DispatchProcess** e enviar-lhe a mensagem **initProcess()** para que este devolva a locação **KHAT:Location** para a qual pode ser despachado **TRAIN.** As colaborações entre objetos que ocasionam esta auto-mensagem são apresentadas na seção 4.3.2.5.

O controlador da simulação prossegue verificando a existência da locação **KHAT.** Se **KHAT** existir, então o controlador envia uma auto-mensagem **trainProceeds()** para deslocar **TRAIN** (ver seção 4.3.2.9). Caso contrário, o controlador envia a auto-mensagem **delayTrainByBlockage()** para atrasar **TRAIN** por um intervalo de tempo (ver seção 4.3.2.10).

Se o tempo de duração da simulação **Config.SIMULATION_TIME** for atingido (cumpre-se a condição **[XL> = Config.SimulationTime]**) dados relacionados com os diferentes tempos dos trens nos segmentos são recolhidos e armazenados. Caso contrário o controlador procura e trata o próximo evento.

O processo descrito anteriormente, bem como o respectivo diagrama, representa somente as colaborações entre classes para realizar uma simulação. Caso sejam necessárias mais de uma simulação (**Config.SIMULATION_NTIMES**) repete-se o processo para as outras simulações.

4.3.2.3. Colaborações entre classes: Perturbações dos trens

Esta seção mostra como as classes colaboram com o controlador da simulação **Controller:SimulationController** para dar resposta à auto-mensagem **doDisruption(disruption).** O diagrama de colaboração é apresentado na Figura 4.15.

O controlador envia mensagens à perturbação **disruption:TrainDisruption** para obter o trem ao qual deve-se perturbar **train:RealTrain**, a hora de início **init** da perturbação e a duração **length** desta.

A seguir o controlador envia mensagens ao trem a perturbar **train:RealTrain** para obter a hora da sua partida **depart** e a hora na qual culminará sua atividade **xnext.** Se o trem já partiu **[depart <init]** e a hora de início da perturbação é menor ou igual que a hora do próximo evento **[init <= xnext],** então o controlador envia a mensagem **disruptTrain(lenght)** ao trem **train:RealTrain** para que este sofra uma perturbação. Esta mensagem altera o momento no qual o trem culminará sua atividade. Finalmente, o controlador notifica ao controlador de resultados **SystemController.REPORT:ReportController** que um trem sofreu uma perturbação.

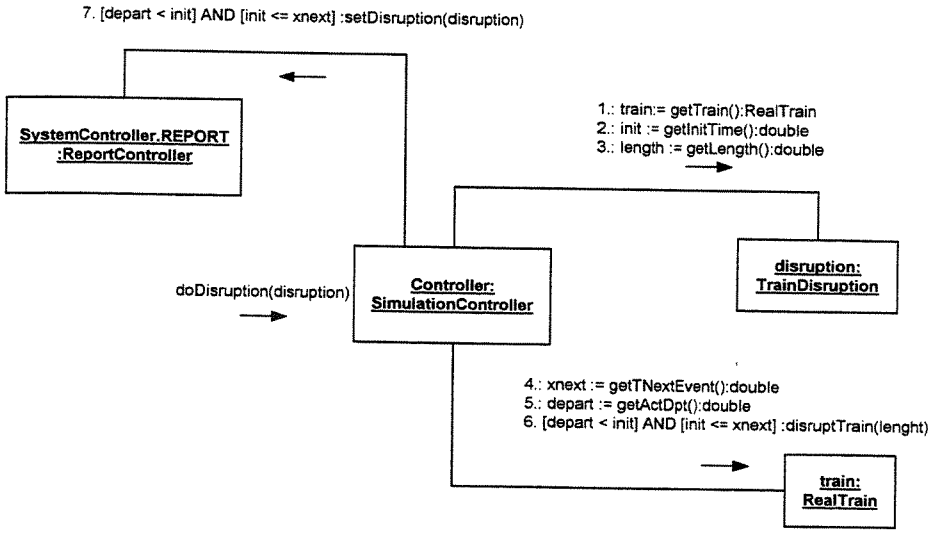


Figura 4.15 Diagrama de colaboração: Perturbações dos trens

4.3.2.4. Colaborações entre classes: Realizar atividades de manutenção

Mostra-se a seguir como as classes colaboram com o controlador da simulação **Controller:SimulationController** para dar resposta à auto-mensagem **doMaintenance().** O diagrama de colaboração é apresentado na Figura 4.16.

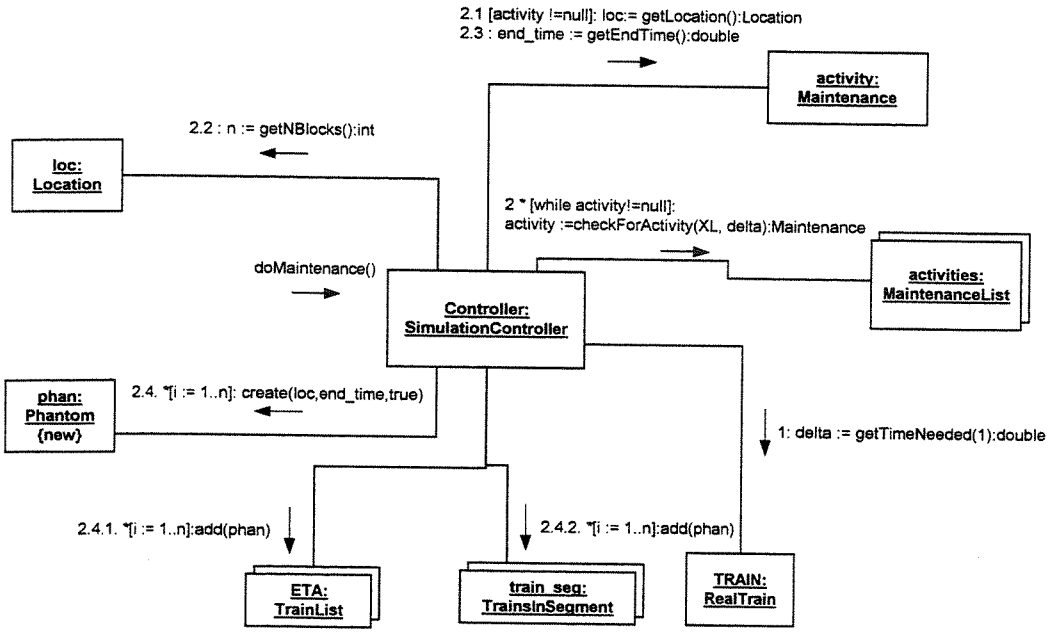


Figura 4.16 Diagrama de colaboração: Realizar atividades de manutenção

O controlador envia a mensagem **delta := getTimeNeeded(1):double** ao trem atual **TRAIN** para determinar o tempo que necessita para percorrer o segmento seguinte de seu percurso.

O controlador envia a mensagem **activity:=checkForActivity(XL,delta): Maintenance** à lista de atividades de manutenção **activities:MaintenanceLista** para determinar as atividades a serem realizadas no intervalo de tempo **[XL, XL+delta]**.

Quando uma atividade é devolvida pela lista, o controlador cria – usando a mensagem **create(loc,end_time,true)** – tantos trens imaginários quantas sinalizações intermediárias existam na locação – **loc:Location** – onde será realizada a atividade. O parâmetro **loc** especifica em que locação é alocado o trem imaginário (onde será realizada a manutenção), o parâmetro, **end_time** indica o momento a partir do qual o trem imaginário deixará de existir (relacionado com o fim da manutenção) e o parâmetro **true** indica que o trem imaginário foi criado por razões de manutenção. O controlador obtém o número de sinalizações intermediárias enviando a mensagem **n := getNBlocks():int** a **loc:Location** e obtém o término da manutenção com a mensagem **end_time:= getEndTime**():**double**. Os trens imaginários criados são inseridos na lista de trens da malha ferroviária (**ETA:TrainList**) e no estado da malha (**train_seg:TrainsInSegment**).

4.3.2.5. Colaborações entre classes: Rotina de despacho

Esta seção mostra como as classes colaboram com a rotina de despacho **disp:DispatchProcess** para responder à mensagem **initProcess()** enviada pelo controlador da simulação e devolver uma locação **KHAT** para onde deslocar o trem **TRAIN**. O diagrama de colaborações é apresentado na Figura 4.17.

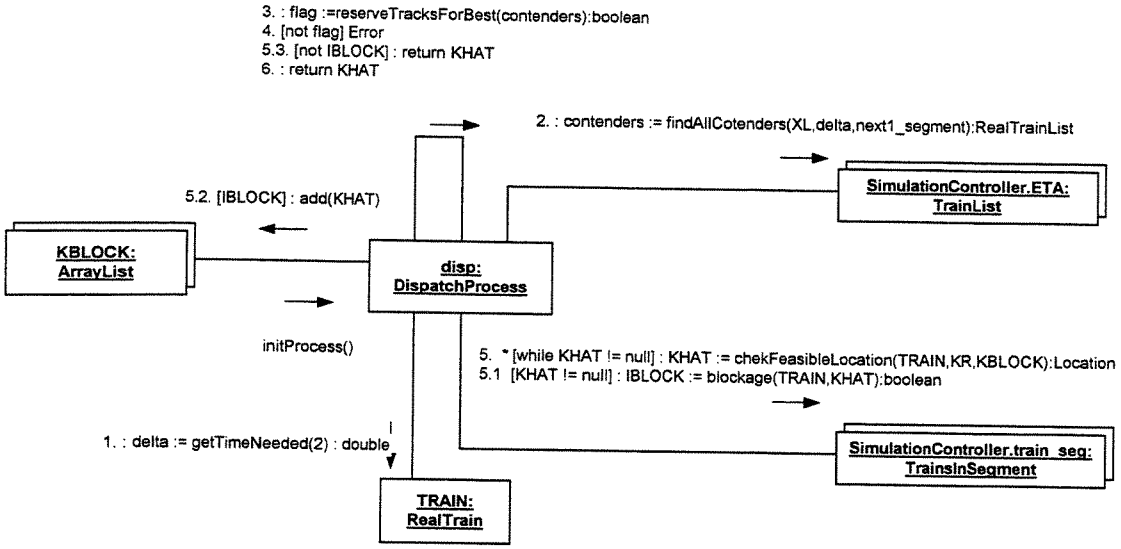


Figura 4.17 Diagrama de colaboração: Rotina de despacho

Para encontrar uma locação para onde deslocar **TRAIN:RealTrain** é necessário determinar primeiro quais trens concorrem pelo uso do mesmo segmento que o trem **TRAIN** tem que se deslocar. A rotina de despacho determina estes trens enviando a mensagem **contenders:= findAllCotenders(XL,delta,next1_segment): RealTrainList** à lista de trens na malha ferroviária **SimulationController.ETA:TrainList.** Onde **next1_segment** é o próximo segmento no percurso do trem. Esta mensagem será analisada na seção 4.3.2.6.

Em geral tem-se que reservar trechos para alguns dos trens concorrentes (**reserveTracksForBest(contenders)**). A escolha de trens que terão trechos reservados depende da função de despacho a otimizar. Esta é definida em um arquivo de texto **Config.OPTIMIZATION_FILE_NAME**, fazendo uso da linguagem desenvolvida (Apêndice). A rotina de despacho **disp:DispatchProcess** cria uma instância da classe que controla a interpretação de arquivos **interprete:InterpreteController** e envia mensagens para que interprete o arquivo de texto e devolva o trem selecionado.

Depois de ter reservado as locações necessárias a rotina de despacho envia a mensagem **KHAT:=checkFeasibleLocation(TRAIN,KR,KBLOCK):Location** ao estado da malha ferroviária para que devolva uma locação (**KHAT**) para a qual o trem **TRAIN** possa ser deslocado. A escolha desta locação descarta as locações reservadas

(**KR**) e as locações que, se fossem usadas pelo trem, ocasionariam bloqueio de linha (**KBLOCK**).

Antes de devolver a locação **KHAT** ao controlador da simulação, tem-se que verificar se o deslocamento do trem **TRAIN** até **KHAT** não ocasiona bloqueio. Para isto, a rotina de despacho envia a mensagem **blockage(TRAIN,KHAT):boolean** ao estado da malha ferroviária. Se o deslocamento causa bloqueio, a rotina de despacho insere **KHAT** na lista de locações que ocasionam bloqueio **KBLOCK:LocationList** e procura por outra locação. Caso contrário, a rotina de despacho devolve **KHAT** ao controlador da simulação.

4.3.2.6. Colaborações entre classes: Trens concorrentes

Esta seção mostra como as classes colaboram com a lista de trens na malha ferroviária - **ETA:TrainList** - para responder à mensagem **contenders:= findAllCotenders(XL,delta,next1_segment):RealTrainList.** Esta mensagem é enviada pela rotina de despacho **disp:DispatchProcess** com a finalidade de obter a lista de trens que estão concorrendo pelo segmento a ser ocupado por **TRAIN**. O diagrama de colaborações é mostrado na Figura 4.18.

A lista de trens na malha procura, entre todos seus elementos, trens que pretendam usar alguma locação do segmento **next_segment1** no intervalo de tempo **[XL,XL+delta].**

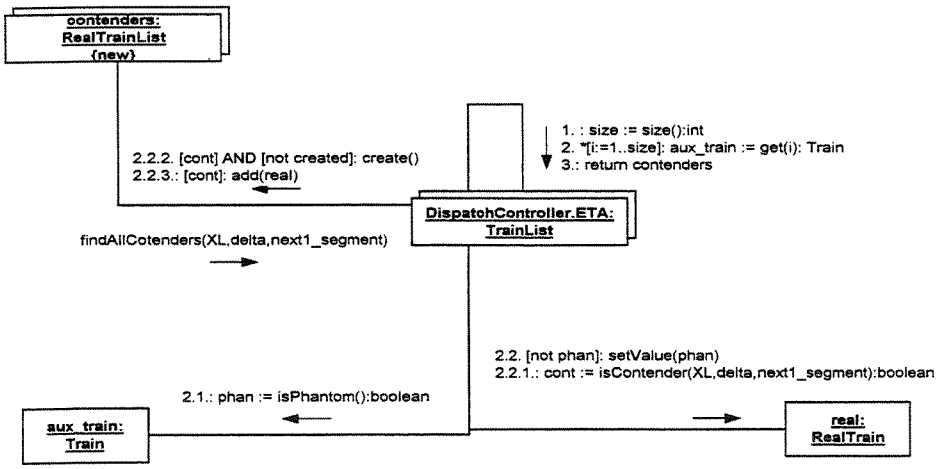


Figura 4.18 Diagrama de colaboração: Trens concorrentes

A lista obtém um dos seus trens usando a mensagem **aux_train:=get(i):Train**. Em seguida, testa se o trem **aux_train** é um trem imaginário usando a mensagem **isPhantom:boolean**. Se for imaginário, procura-se por outro trem. Caso contrário, o trem **real:RealTrain** assume o valor de **aux_train**.

A lista envia a mensagem **isContender(XL,delta,next1_segment):boolean** ao trem **real:RealTrain** para verificar se este pretende usar o segmento **next1_segment** no intervalo de tempo **[XL,XL+delta]**. Pode obter uma resposta positiva ou negativa. Se a resposta for positiva, a lista manda inserir o trem dentro da lista de trens concorrentes **contenders:RealTrainList**.

O diagrama de colaboração da mensagem **isContender (XL,delta,next1_segment):boolean** é apresentado na Figura 4.19. Este diagrama mostra que, se o trem ainda não partiu ou já passou o segmento, então ele não é um trem concorrente. Obtém-se a hora na qual o trem chega ao segmento e verifica-se se está dentro do respectivo intervalo de tempo.

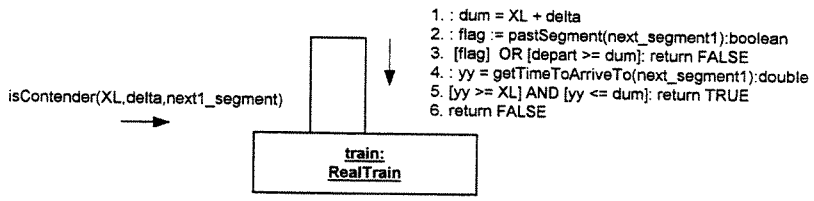


Figura 4.19 Diagrama de colaboração: Trem concorrente

4.3.2.7. Colaborações entre classes: Locação disponível

Esta seção discute como as classes colaboram com o estado da malha ferroviária **SimulationController.train_seg:TrainsInSegment** para responder à mensagem **KHAT:=checkFeasibleLocation(TRAIN,KR,KBLOCK):Location,** enviada pela rotina de despacho **disp:DispatchProcess,** visando devolver uma locação **KHAT** para onde deslocar **TRAIN**. O diagrama de colaboração é apresentado na Figura 4.20.

SimulationController.train_seg envia a mensagem **segment1:= getSegment(1):Segment** a **TRAIN** para obter o segmento seguinte do seu

percurso. A seguir, do estado da malha obtém-se o conjunto de locações disponíveis **KC:LocationList** através da auto-mensagem **feasible (TRAIN,KR,KBLOCK):LocationList**. Esta mensagem será analisada posteriormente.

Continuando, o estado da malha obtém o número de elementos **size:size():int** da lista de locações disponíveis. Se o número de elementos for igual a zero então o estado da malha ferroviária devolve o valor de **null** à rotina de despacho **disp:DispatchProcess** significando que não foi encontrada nenhuma locação disponível. Se o número de elementos for maior que zero, então o estado da malha ferroviária envia a mensagem **stopsIn(segment1):boolean** ao **TRAIN** despachado, para verificar se este fará escala no segmento **segment1**. Se este for o caso, então o estado da malha obtém, da lista de locações disponíveis, aquela com maior tempo de percurso usando a mensagem **KHAT:= getSlowestLocation():Location**. Caso contrário, obtém da lista de locações disponíveis aquela com o menor tempo de percurso usando a mensagem **KHAT:= getFastestLocation():Location**. O estado da malha devolve **KHAT** à rotina de despacho **disp:DispatchProcess**.

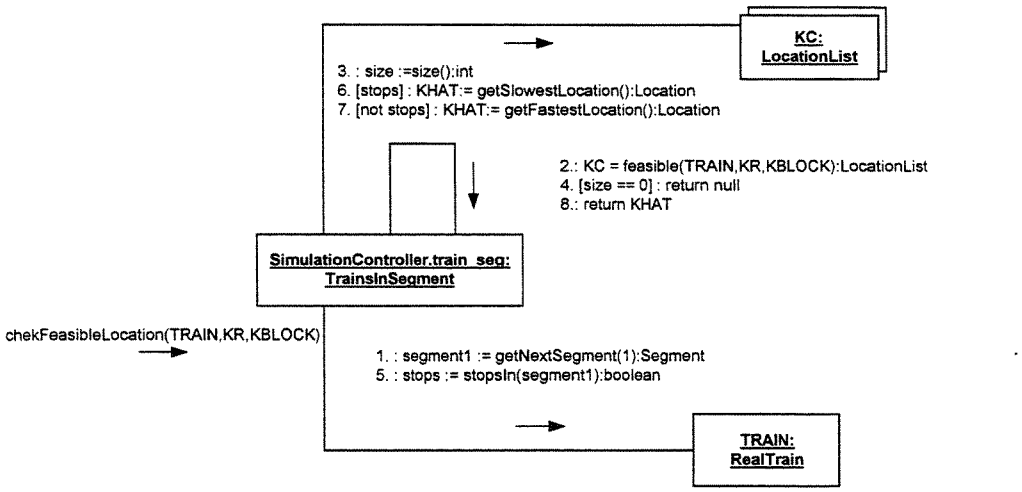


Figura 4.20 Diagrama de colaboração: Locação disponível

A auto-mensagem **feasible(TRAIN,KR,KBLOCK)** do estado da malha obtém um conjunto de locações disponíveis. O diagrama de colaboração desta auto-mensagem é mostrado na Figura 4.21. Para obter este conjunto de locações, o estado da malha ferroviária cria primeiro um conjunto de locações que não serão

consideradas **discard:LocationList** e insere neste conjunto as locações reservadas **KR** e as que ocasionam bloqueio **KBLOCK**.

A seguir, o estado da malha obtém do **TRAIN:RealTrain** o próximo segmento **segment1:Segment** do seu percurso, seu sentido atual de viagem **ndir** e seu modelo de percurso **journey:JourneyPattern**. O estado da malha envia então a auto-mensagem **SET:=discardLocations(segment1,ndir):LocationList** para obter um outro conjunto de locações não consideradas. Este conjunto é formado pelas locações ocupadas por trens viajando no sentido oposto e pelas locações que estão cheias.

Em seguida, o estado da malha ferroviária envia a mensagem **KB:=getFeasiblesTracks(segment1,ndir,discard):LocationList** à locação atual do trem **current:Location**. Esta mensagem obtém a lista de locações do segmento **segment1** aquelas que são fisicamente acessíveis desde a locação atual do trem **current**. Finalmente, o estado da malha remove **SET** de **KB** e retorna o restante de **KB**.

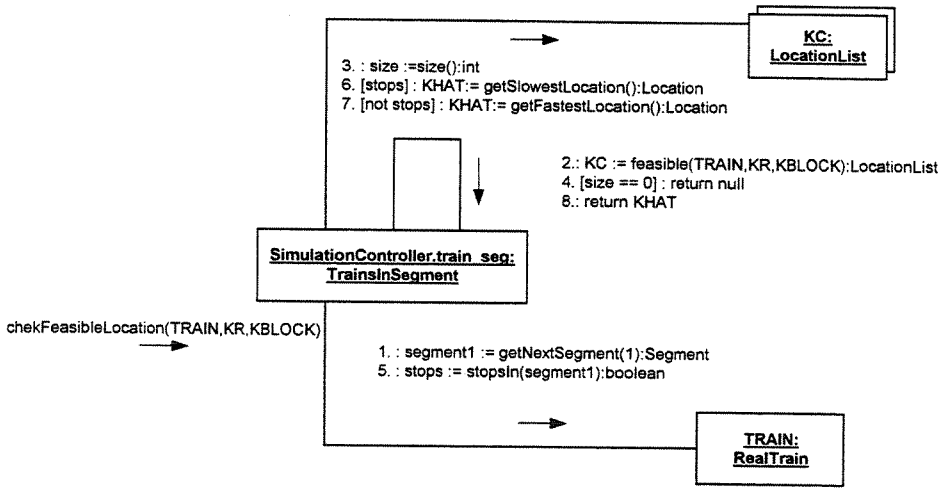


Figura 4.21 Diagrama de colaboração: Locações disponíveis

4.3.2.8. Colaborações entre classes: Prevenção de bloqueio na malha ferroviária

Discute-se aqui como as classes colaboram com o estado da malha ferroviária **SimulationController.train_seg:TrainsInSegment** para responder à mensagem **blockage(TRAIN,KHAT):boolean** enviada pela rotina de despacho **disp.DispathProcess** visando detectar se o deslocamento de um trem **TRAIN** a uma locação **KHAT** originará um bloqueio. O diagrama de colaboração é apresentado na Figura 4.22.

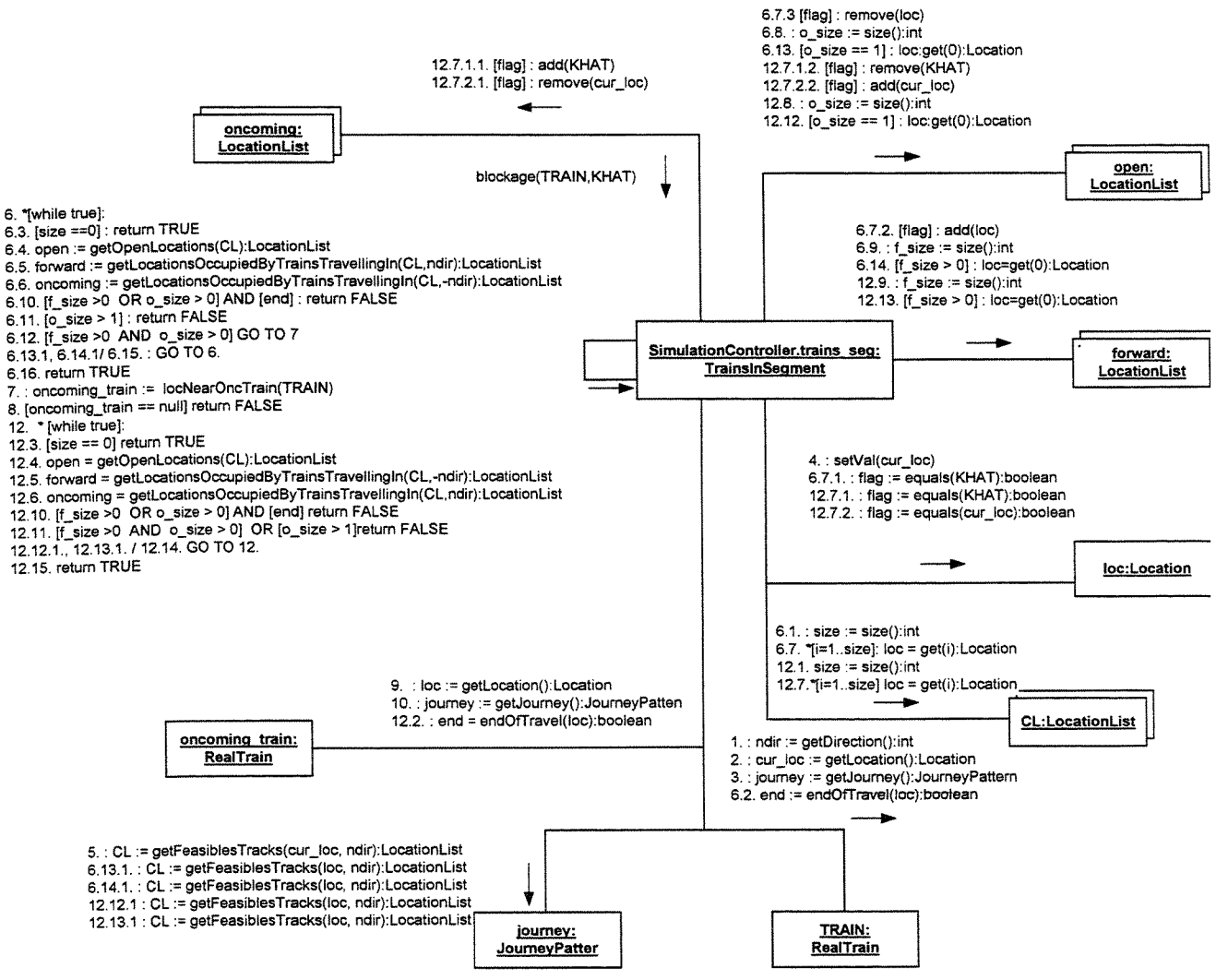


Figura 4.22 Diagrama de colaboração: Prevenção de bloqueio na malha

A idéia central desta atividade é encontrar um *segmento livre* para o trem **TRAIN** e, se for necessário, encontrar um *segmento livre* para a o *trem contrário* mais próximo.

O **SimulationController.trains_seg** envia mensagens ao trem **TRAIN:RealTrain** para obter seu sentido de viagem **ndir,** a locação que está ocupando **cur_loc** e seu modelo de percurso **journey:JourneyPattern**. Atribui-se à locação **loc:Location** o valor de **cur_loc**. Posteriormente, envia-se a mensagem **CL:=getFeasiblesTracks(cur_loc,ndir): LocationList** ao modelo de percurso, para obter o conjunto de locações **CL:LocationList** fisicamente acessíveis a partir da posição atual do **TRAIN** e que pertençam ao percurso do trem.

O processo termina quando um *segmento livre* for encontrado para **TRAIN** ou quando não for possível encontrar um *segmento livre* para este trem. Se não existir nenhuma locação no conjunto **CL** então o algoritmo já chegou ao final do percurso de **TRAIN** e seu deslocamento para a locação **KHAT** não ocasionará bloqueio.

Caso exista alguma locação em **CL,** obtém-se a lista de locações abertas usando a mensagem **open:=getOpenLocations(CL):LocationList**, a lista de locações ocupadas por trens viajando no mesmo sentido que **TRAIN** usando a auto-mensagem **forward:=getLocationsOccupiedByTrainsTravellingIn (CL,ndir):LocationList** e a lista de locações ocupadas por trens viajando no sentido oposto usando a auto-mensagem **oncoming:=getLocationsOccupiedByTrainsTravellingIn(CL,-ndir): LocationList**. O passo seguinte é testar se o conjunto de locações **CL** contém a locação **KHAT**. Se o conjunto conter a locação, então ela é removida da lista de locações **open** e inserida na lista de locações **forward**. Depois, o estado da malha obtém o número de locações **o_size** em **open** e o número de locações **f_size** em **forward**. Se existir locações abertas ou locações ocupadas por trens viajando no mesmo sentido e se tiver chegado ao final do caminho (**[f_size >0 OR o_size > 0] AND [end]**) então encontrou-se um *segmento livre* para **TRAIN**. O mesmo acontece se existir mais de uma locação aberta **[o_size > 1].** Em ambos casos, não é necessário procurar um *segmento livre* para o *trem contrário* mais próximo para garantir que não exista bloqueio na malha.

Se existir alguma locação aberta e alguma locação ocupada por trens viajando no mesmo sentido **[f_size >0 AND o_size > 0]** então encontrou-se o *segmento livre* para **TRAIN** e é necessário procurar um *segmento livre* para o *trem contrário* mais próximo.

Quando existir uma locação **loc:Location** aberta (cumpre-se condição **[o_size == 1]**), cria-se a partir dela o conjunto de locações **CL** enviando ao modelo de percurso a mensagem **CL:=getFeasiblesTracks(loc,ndir): LocationList.** Repete-se o processo descrito considerando o novo conjunto de locações acessíveis **CL**.

Caso exista pelo menos uma locação **loc:Location** contendo trens viajando no mesmo sentido de **TRAIN** (cumpre-se condição **[f_size > 0]**), cria-se, a partir de uma delas, o conjunto de locações **CL** enviando ao modelo de percurso a mensagem **CL:=getFeasiblesTracks(loc,ndir):LocationList.** Repete-se o processo descrito considerando o novo conjunto de locações acessíveis **CL**.

Quando não existir locações nem no conjunto **open** e nem **forward** ([**f_size == 0 AND o_size == 0**]) não é possível encontrar o *segmento livre* para **TRAIN** e seu deslocamento a **KHAT** produz bloqueio.

Como foi mencionado em parágrafos anteriores, as vezes é necessário encontrar um *segmento livre* para o trem oposto. A auto-mensagem **oncoming_train:= locNearOncTrain(TRAIN):RealTrain** fornece o trem oposto mais próximo a **TRAIN**. O processo para encontrar um *segmento livre* para este trem é similar ao processo anteriormente descrito. Se for encontrado um *segmento livre* para este trem, então o deslocamento de **TRAIN** a **KHAT** não ocasiona bloqueio na malha.

4.3.2.9. Colaboração entre classes: Deslocar o trem

Discute-se agora como as classes colaboram com o controlador da simulação **Controller:SimulationController** para responder à auto-mensagem **trainProceeds()** para deslocar **TRAIN** para **KHAT**. O diagrama de colaboração é mostrado na Figura 4.23.

A lógica desta atividade é a seguinte: o controlador da simulação cria um trem imaginário que é inserido na lista de trens na malha **ETA:TrainList** e no estado da

malha ferroviária **train_seg:TrainsInSegment**. Este trem imaginário é alocado na locação **last_location** que representa a locação atual do **TRAIN.**

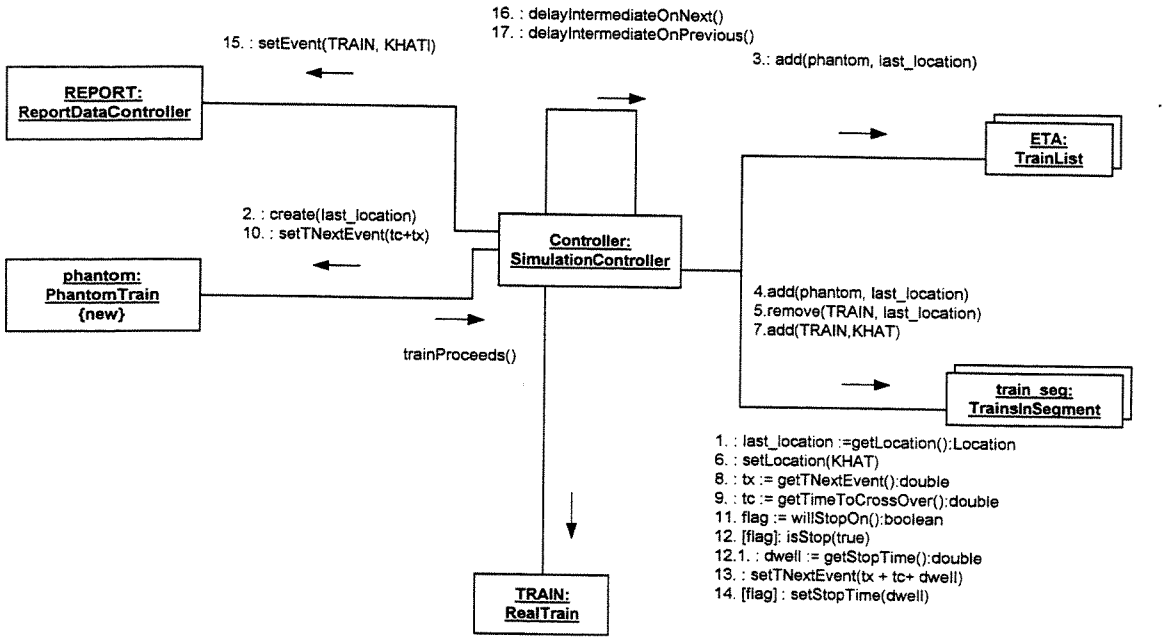


Figura 4.23 Diagrama de colaboração: Deslocar o trem

O controlador da simulação aloca **TRAIN** em **KHAT** e atualiza sua posição em **train_seg:TrainsInSegment**. Também envia a mensagem **setTNextEvent (tx+tc+dwell)** a **TRAIN** para que atualize o instante de seu próximo evento.

O controlador da simulação comunica o deslocamento ao controlador de dados de saída **REPORT:ReportDataController**. O controlador da simulação, também atualiza os instantes dos próximos eventos correspondentes aos trens alocados na antiga locação (**last_location)** do **TRAIN** ou de trens alocados em **KHAT** enviando as auto-mensagens **delayIntermediateOnPrevious()** e **delayIntermediateOnNext()**.

4.3.2.10. Colaboração entre classes: Atrasar o trem

Mostra-se aqui como as classes colaboram com o controlador da simulação **Controller:SimulationController** para responder à auto-mensagem **delayByBlockage()** para atrasar **TRAIN** na sua locação atual. O diagrama de colaboração é apresentado na Figura 4.24.

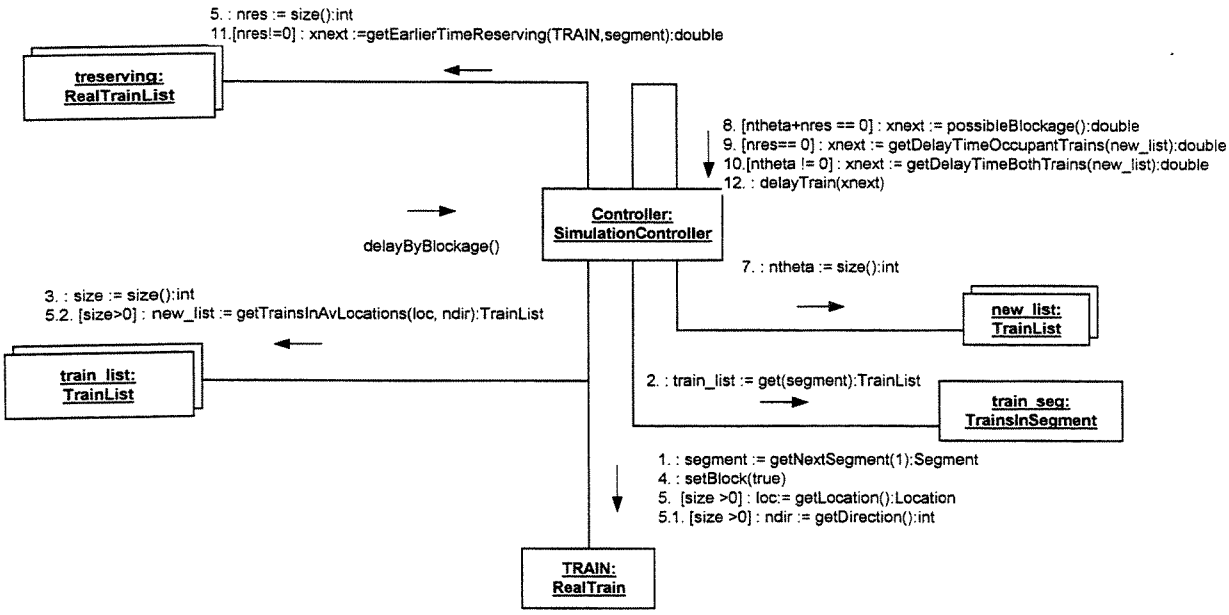


Figura 4.24 Diagrama de colaboração: Atrasar o trem

O controlador obtém o segmento seguinte **segment:Segment** do percurso do trem **TRAIN:RealTrain** e a lista de trens alocados neste segmento **train_list:TrainList**.

A seguir, se **train_list** contém algum elemento, o controlador obtém a locação atual do trem **loc:Location** e o sentido de viagem **ndir** e envia a mensagem **new_list:= getTrainsInAvLocations(loc, ndir):TrainList** a **train_list** para obter a lista de trens que no futuro podem desocupar as locações que podem ser usadas por **TRAIN.**

Se não existir nenhum trem em **new_list:TrainList** e não existir nenhum trem em **treserving:RealTrainList** (trens reservando locações), então o tempo de atraso do trem é obtido através da auto-mensagem do controlador da simulação **xnext:= possibleBlockage():double**. Basicamente esta mensagem procura os trens próximos viajando no sentido oposto e tenta encontrar uma locação para despachá-los. Se algum deles puder ser despachado o **TRAIN** será atrasado até o próximo evento deste trem, caso contrário é atrasado até que o próximo evento aconteça.

Se não existirem trens reservando locações, o tempo de atraso será obtido através da auto-mensagem do controlador da simulação **xnext :=**

getDelayTimeOccupantTrains(new_list):double. Esta mensagem procura na lista **new_list** trens que sejam imaginários ou que chegaram a seu destino, ou que possam ser despachados. Se algum destes trens for encontrado, o **TRAIN** será atrasado até o próximo evento desse trem.

Se existirem elementos só na lista **treserving:RealTrainList**, o controlador envia para esta lista a mensagem **xnext:=getEarlierTimeReserving (TRAIN,segment):double**, onde **xnext** é o tempo de atraso de **TRAIN**. Esta mensagem obtém o menor tempo no qual um trem da lista chegará (se viaja no mesmo sentido) ou passará (se viaja no sentido oposto) pelo segmento **segment:Segment**.

Se existirem elementos nas duas listas (**new_list** e **treserving**) o tempo de atraso do trem será obtido através da auto-mensagem do controlador **xnext:=getDelayTimeBothTrains(new_list):double**. Esta mensagem devolve o menor dos tempos fornecidos pelas mensagens **treserving.getEarlierTimeReserving(TRAIN,segment)** e **getDelayTimeOccupantTrains(new_list).**

A auto-mensagem **delayTrain(xnext)** atualiza o próximo evento de **TRAIN (setTNextEvent(xnext))** e atualiza os dados para relatório.

A partir dos diagramas de colaborações apresentados podem ser extraídas as principais operações das classes do modelo.

4.4. Resumo

Neste capítulo tratou-se da modelagem orientada a objetos do sistema computacional baseado no modelo de linha proposto.

O sistema foi descrito usando a visão Caso de Uso e Lógica do Sistema proposta por [BOO98].

A primeira visão foi feita utilizando diagramas de casos de uso. Esta visão descreve o sistema como percebido pelo usuário. O sistema computacional apresenta quatro casos de uso. O primeiro deles é Atualizar Dados de Simulação, o qual refere-se a permitir ao usuário definir os componentes do modelo de linha e salvar ou abrir arquivos contendo estas

informações. O segundo caso de uso refere-se à funcionalidade que permite ao usuário configurar parâmetros da simulação e dar início à(s) simulação(es). O terceiro trata de como mostrar os resultados obtidos aos usuários e permitir o armazenamento dos resultados. E o último, refere-se a permitir ao usuário manipular arquivos de funções de otimização.

A segunda visão descreve a estrutura estática e dinâmica do sistema. Para modelar a estrutura estática foram usados diagramas de classes. Nestes diagramas podem ser observadas as classes criadas para modelar os componentes do modelo de linha proposto e a sua lógica. A dinâmica do modelo de linha foi apresentada fazendo uso de diagramas de colaborações. Criou-se diagramas de colaboração para cada um dos passos envolvidos na lógica do modelo.

O capítulo seguinte trata da ferramenta computacional desenvolvida e mostra exemplos de uso da ferramenta.

Capítulo 5

Ferramenta Computacional e Aplicações

O modelo de linha ferroviária proposto foi desenvolvido utilizando, por sua vez, um modelo de sistema com orientação a objetos. A partir desta modelagem implementou-se uma ferramenta computacional. Esta implementação foi realizada usando a linguagem de programação Java. Uma descrição geral do sistema desenvolvido pode ser encontrada em [RON00d].

Este capítulo descreve a ferramenta computacional implementada e apresenta um conjunto de exemplos com o objetivo de ilustrar as aplicações da mesa.

5.1. Descrição da Ferramenta Computacional

Esta seção descreve a ferramenta computacional implementada a partir do modelo de linha proposto. A ferramenta é descrito através das suas janelas de interface com o usuário. Estas janelas mostram um exemplo baseado em dados fornecidos pela NOVOESTE e do seu ramal de PONTA PORÃ. A NOVOESTE atravessa os estados de São Paulo e Mato Grosso do Sul. O ramal PONTA PORÃ está localizado no estado de Mato Grosso do Sul.

No capítulo anterior foram definidos quatro casos de uso que representam as funcionalidades do sistema: **Atualizar Dados de Simulação**, **Despachar Trens**, **Reportar Informação** e **Atualizar Funções de Otimização**. A janela inicial do sistema, Figura 5.1, introduz estas quatro funcionalidades. Como pode ser observado, esta janela possui um menu, uma barra de ferramentas e um painel tabulado (*tabbed pane*) composto por quatro painéis: **Inserção de Dados**, **Simulações**, **Resultados das Simulações** e **Função de Otimização**. Estes painéis estão associados com às funcionalidades do sistema. O menu e a barra de ferramentas possibilitam ao usuário manipular (abrir e salvar) arquivos de dados, arquivos de otimização e arquivos de resultados. Nas subseções seguintes apresenta-se a implementação das quatro funcionalidades do sistema.

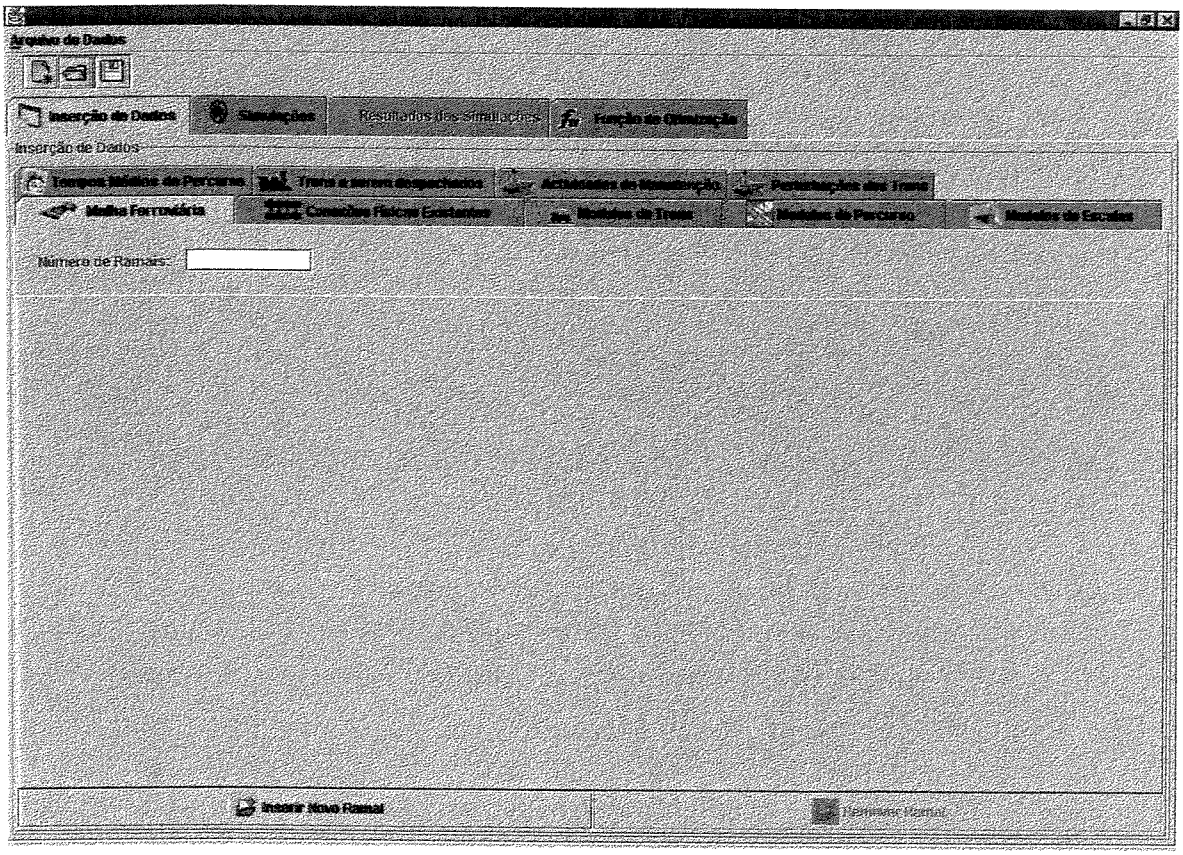


Figura 5.1 Janela: Inicial do sistema

5.1.1. Interface com o usuário: Atualizar Dados de Simulação

Esta seção descreve como o sistema implementa a funcionalidade de **Atualizar dados da simulação**, a qual está relacionada com o painel **Inserção de Dados**. Como foi mencionado no capítulo anterior, esta funcionalidade está subdivida em: Salvar e Abrir arquivos de dados e Atualizar dados dependentes e não dependentes.

O sistema fornece a possibilidade de abrir e salvar arquivos de dados da simulação através do uso do menu ou da barra de ferramentas. Este tipo de arquivos tem associados a ele a extensão "dat". A Figura 5.2 mostra a caixa de diálogo que aparece quando o usuário deseja abrir um arquivo de dados. A caixa de diálogo que aparece quando um usuário deseja salvar um arquivo é similar a esta.

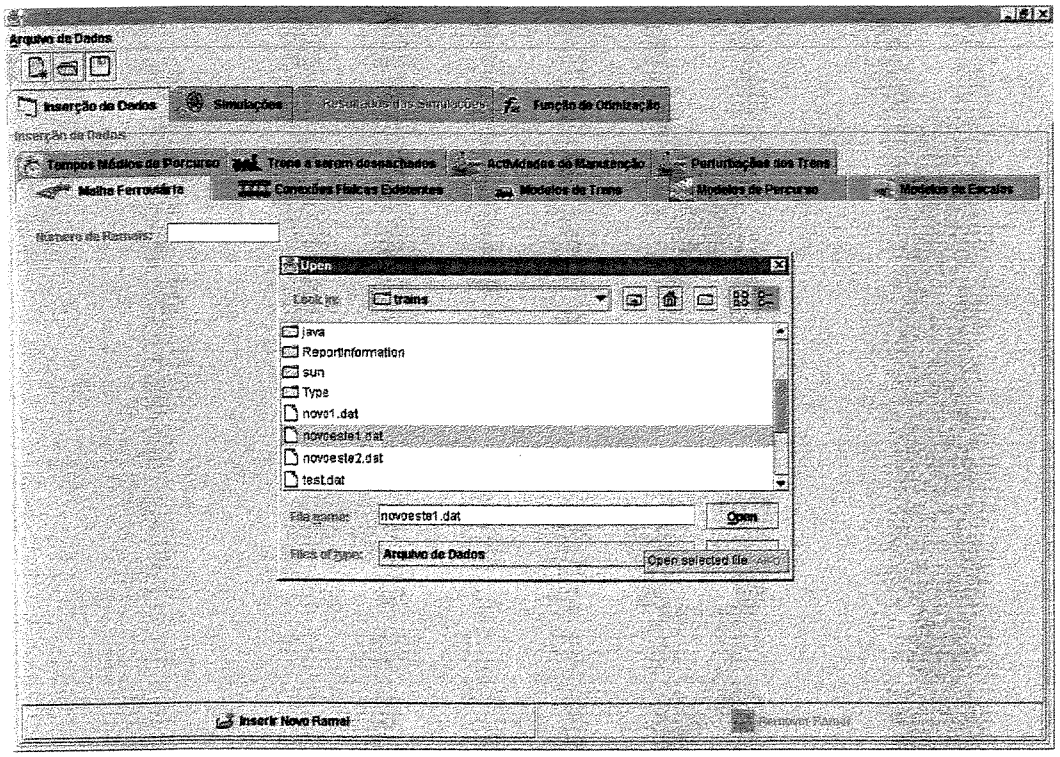


Figura 5.2 Janela: Abrir arquivo de dados

Apesar das funcionalidades **Atualizar dados dependentes** e **Atualizar dados não dependentes** tenham sido definidas, estas são abstratas e portanto não são implementadas. Entretanto, são implementadas as funcionalidades que mantém com elas uma relação de *uso*. No painel **Inserção de Dados** da Figura 5.1, observa-se outro painel tabulado que contém mais nove painéis. Estes nove painéis implementam estas funcionalidades. A seguir detalha-se a implementação destas funcionalidades.

5.1.1.1. Atualização de dados: Malha ferroviária

Os dados da malha ferroviária são dados não dependentes. A inserção destes dados não depende da existência prévia de nenhum outro dado.

A Figura 5.3 mostra a janela de atualização destes dados. A janela **Malha Ferroviária** permite ao usuário especificar quantos ramais (linhas) compõem a malha ferroviária. Quando se trata de uma linha ferroviária simples, o número de ramais da malha é 1. A inserção do número de ramais gera um painel tabulado contendo painéis para permitir a atualização dos dados de cada um dos ramais. No exemplo da Figura 5.3, a malha ferroviária é formada por dois ramais. Outra forma

de inserção ou remoção de ramais da malha ferroviária é através do uso dos botões **Inserir Novo Ramal** e **Remover Ramal**. Estes botões aparecem na maioria de janelas de atualização de dados, tendo em todas as mesmas funcionalidades (inserir ou remover objetos).

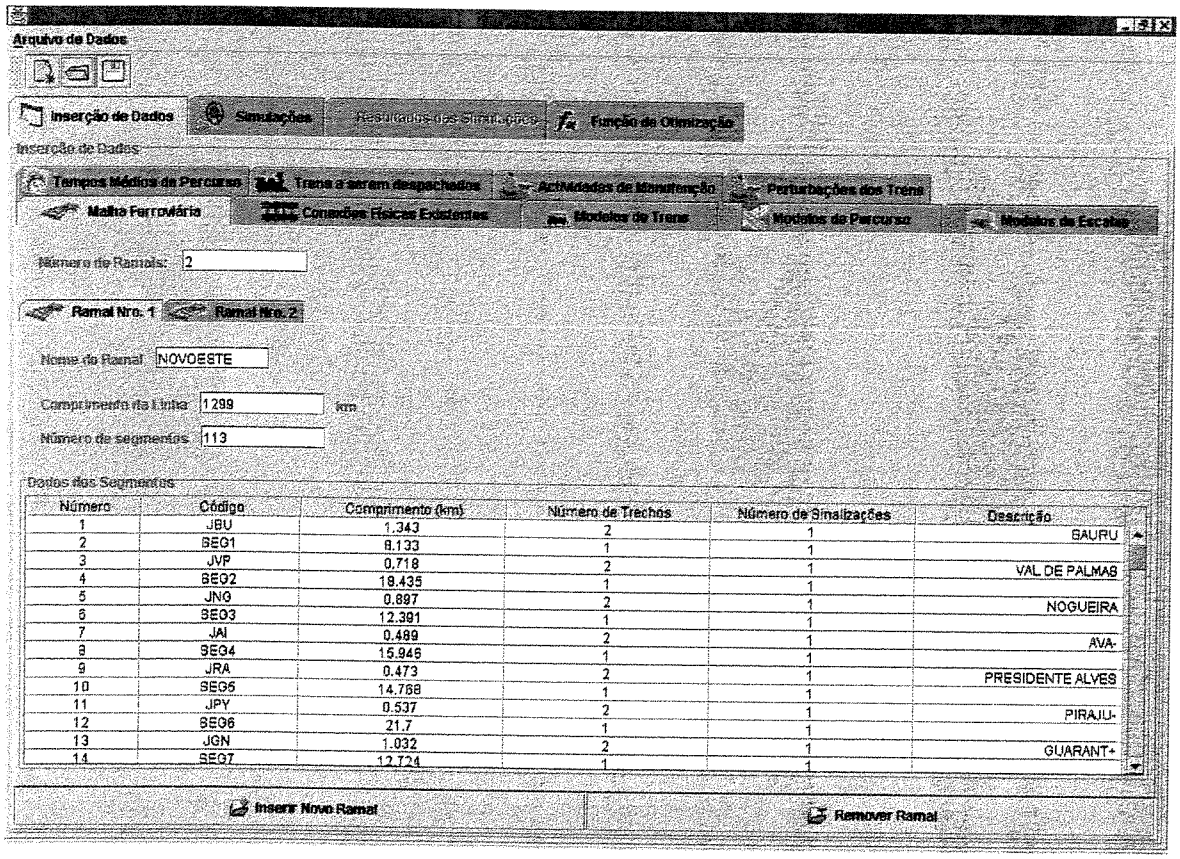


Figura 5.3 Janela: Dados da malha ferroviária

Observa-se também na Figura 5.3 a descrição dos dados de um ramal da malha ferroviária. Este ramal tem o nome NOVOESTE, e 1299 km de comprimento e está composto por 113 segmentos. Estes 113 segmentos estão descritos em uma tabela de dados. Cada um deles tem um número; um código; um comprimento; um número de trechos paralelos; um número de sinalizações intermediárias e uma descrição. O segundo ramal, cuja janela não é mostrada, tem por nome PONTA PORÃ, com 304 km de comprimento e está composto por dez segmentos.

5.1.1.2. Atualização de dados: Conexões físicas existentes

A janela **Conexões Físicas Existentes** (Figura 5.4) implementa a funcionalidade **Atualizar dados de conexões físicas na malha ferroviária**. As conexões físicas referem-se tanto a segmentos consecutivos em um mesmo ramal quanto a conexões físicas entre segmentos de ramais diferentes. Esta janela contém um painel tabulado com três painéis: os dois primeiros permitem ao usuário conectar ou desconectar trechos consecutivos nos ramais NOVOESTE e PONTA PORÃ, o terceiro painel permite definir as conexões físicas entre ambos os ramais.

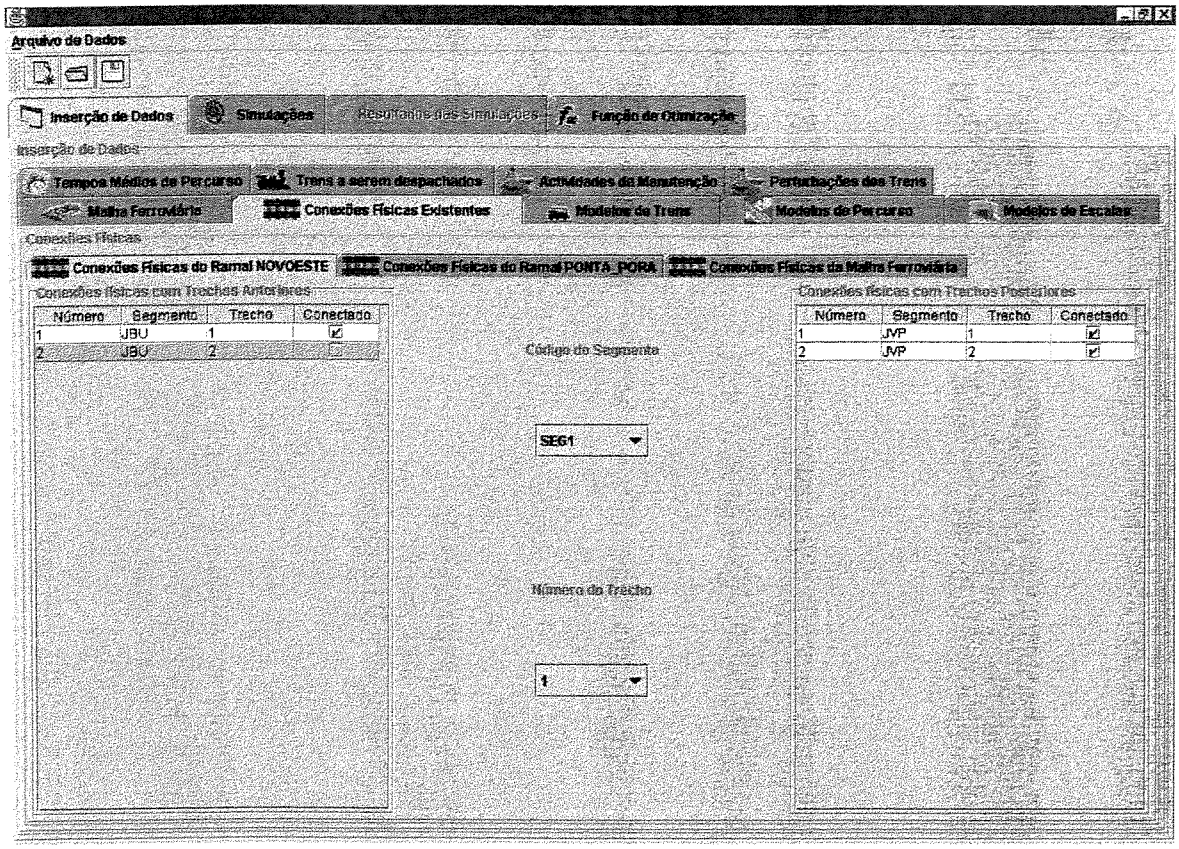


Figura 5.4 Janela: Conexões físicas existentes

A Figura 5.4 mostra a janela **Conexões Físicas do Ramal NOVOESTE.** No centro desta janela aparecem duas *combo boxes* que permitem escolher o segmento e o trecho para os quais se deseja estabelecer conexões. No lado direito aparece uma tabela que contém os dados do segmento posterior ao segmento escolhido e no lado esquerdo uma tabela com os dados do segmento anterior. Além dos dados dos segmentos anterior e posterior, as tabelas contém informação relacionada à conexão

física. O conteúdo destas tabelas variam conforme o segmento ou trecho selecionado. No exemplo da Figura 5.4, vemos que antes do segmento SEG1 vem o segmento JBU e que depois de SEG1 vem o segmento JVP. O trecho 1 do segmento SEG1 está conectado com o trecho 1 de JBU mas não com o trecho 2 e está conectado com os dois trechos de JVP.

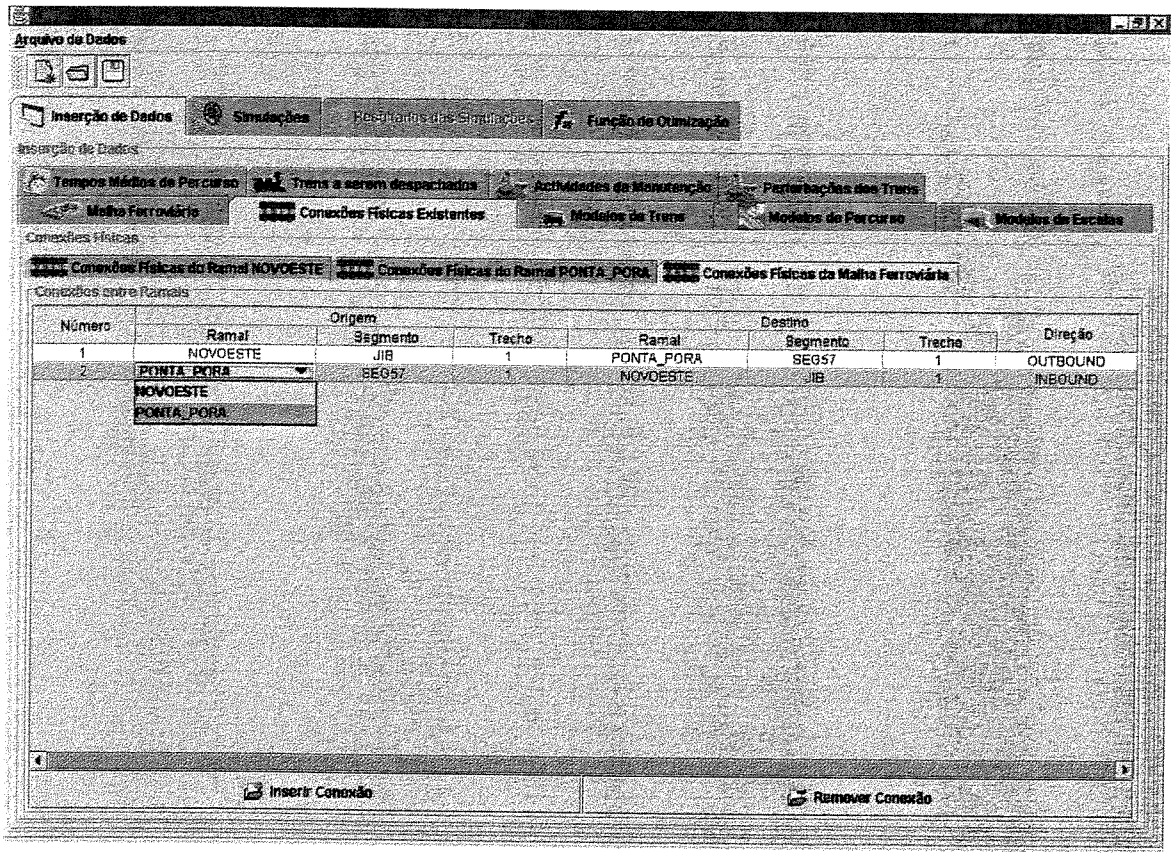


Figura 5.5 Janela: Conexões físicas da malha ferroviária

A Figura 5.5 mostra a janela **Conexões Físicas da Malha Ferroviária**. Esta janela, permite definir as conexões físicas entre os ramais. Para inserir ou remover uma conexão utilizam-se os botões **Inserir Conexão** e **Remover Conexão**. Na primeira conexão da Figura 5.5, podemos observar que o trecho 1 do segmento JIB do ramal NOVOESTE está conectado com o trecho 1 do segmento SEG57 do ramal PONTA PORÃ no sentido *outbound.*

Além da Figura 5.5 mostrar a definição de conexões físicas entre ramais, ela também permite observar a dependência entre dados. A definição das conexões físicas permissíveis depende da definição prévia dos dados da linha ferroviária. Esta

figura foi armazenada quando o usuário está definindo o nome do ramal que será conectado com outro ramal. Nesse momento surge uma lista com os nomes dos ramais existentes para que o usuário escolha um deles. Esta lista é mostrada com o propósito de facilitar a atualização de dados e evitar inconsistência nos mesmos.

Se o usuário tentar acessar a uma janela que atualiza dados dependentes e os dados dos quais depende ainda não foram inseridos, o sistema envia uma mensagem de advertência e o orienta para atualizar os dados que estão ausentes. A Figura 5.6 ilustra uma mensagem de erro que ocorre quando um usuário tenta acessar a janela de atualização de dados de perturbações de trens sem ter inserido os dados dos trens anteriormente.

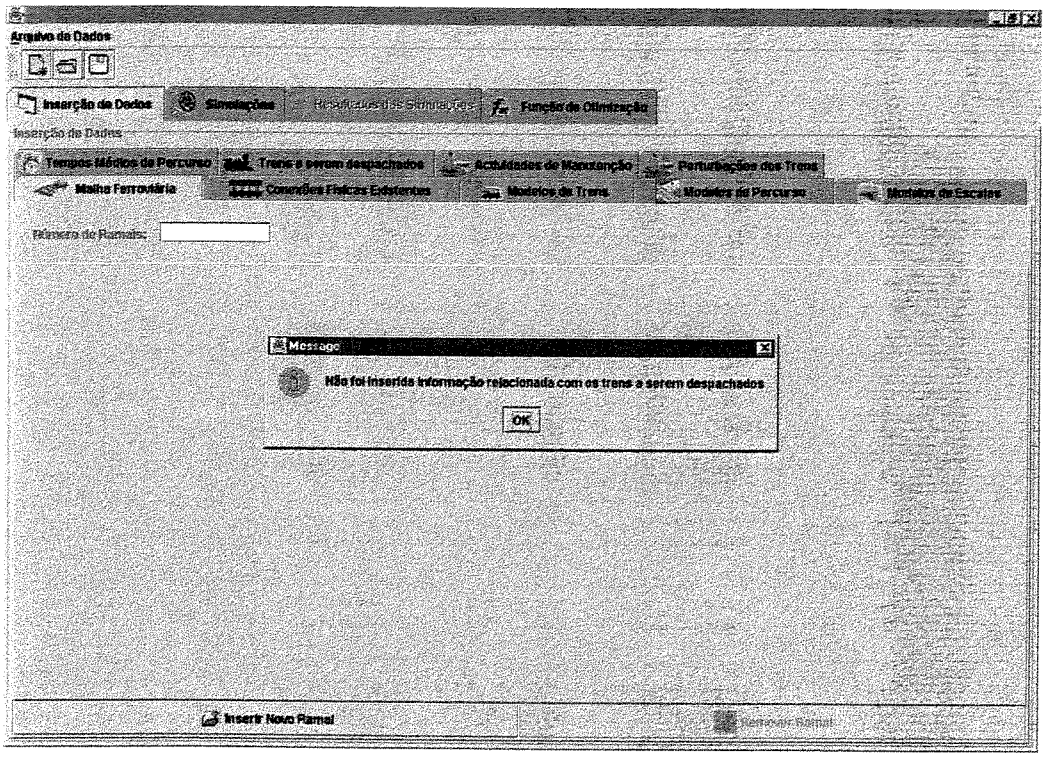


Figura 5.6 Janela: Mensagem de erro

5.1.1.3. Atualização de dados: Modelos de trens

A janela **Modelos de Trens** (Figura 5.7) mostra a implementação das funcionalidades **Atualizar dados dos modelos de trens não dependentes** e **Atualizar dados dos modelos de trens dependentes**. Esta janela contém um painel tabulado com dois painéis, cada um deles associado a uma funcionalidade.

Estes painéis chamam-se: **Modelos de trens não dependentes** e **Modelos de trens dependentes.**

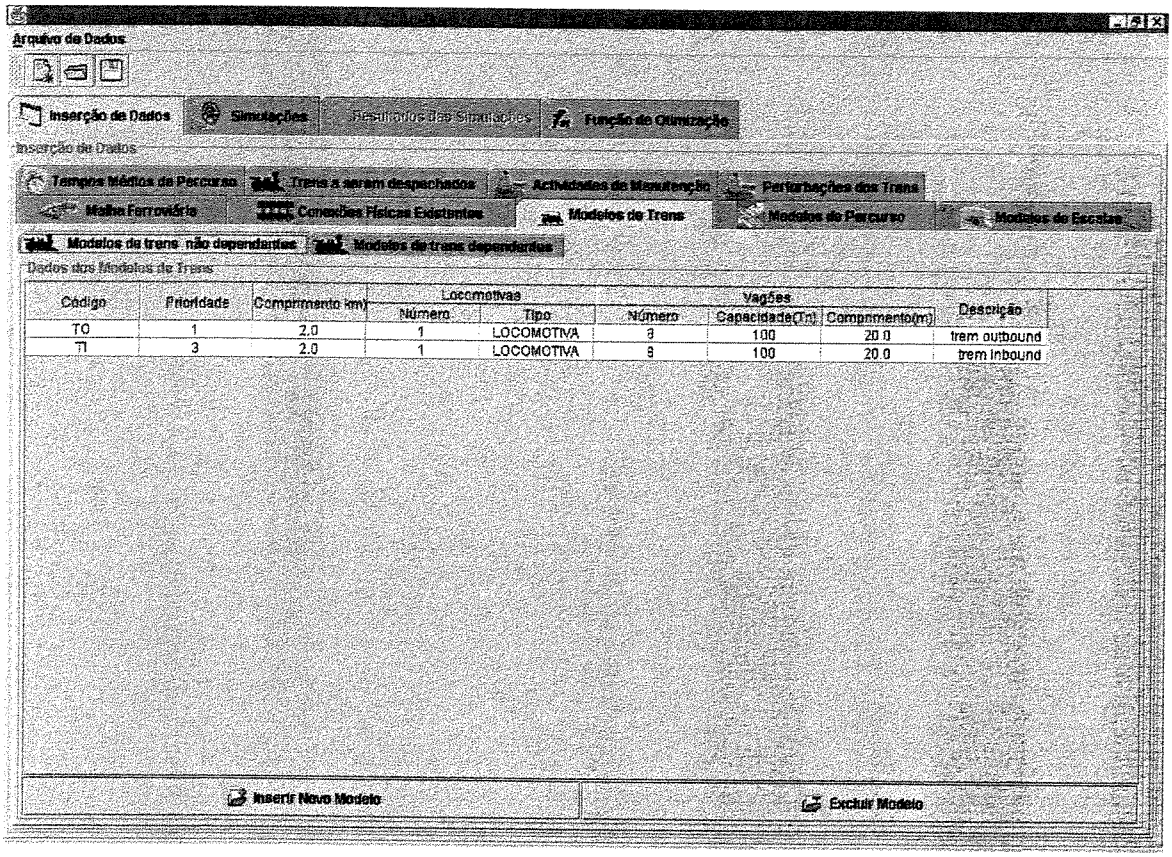


Figura 5.7 Janela: Modelos de trens não dependentes

O painel **Modelos de trens não dependentes** contém uma tabela que apresenta os dados deste tipo de modelos de trens (Figura 5.7). Para cada modelo de trem define-se: um código; uma prioridade; um comprimento; um número de locomotivas e o tipo delas; um número de vagões, a capacidade e comprimento de cada vagão e por último uma descrição do trem. Na Figura 5.7, observa-se dois modelos de trens não dependentes: TO e TI, respectivamente.

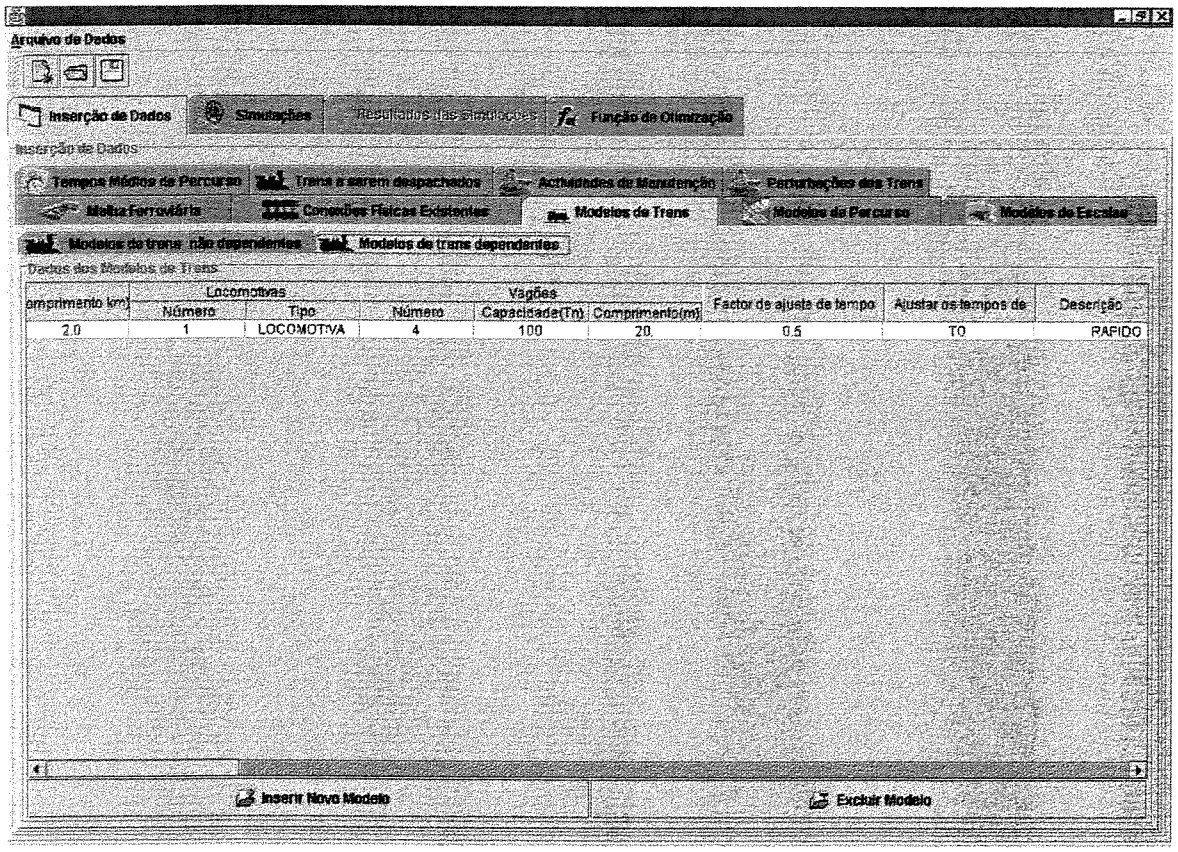


Figura 5.8 Janela: Modelos de trens dependentes

A Figura 5.8 mostra a janela **Modelos de trens dependentes.** Esta janela contém uma tabela com os dados dos modelos de trens dependentes. Para cada modelo de trem define-se: um código; uma prioridade; um comprimento; um número de locomotivas e o tipo delas; um número de vagões, a capacidade e comprimento de cada vagão; um fator de ajuste do tempo de percurso; o código do modelo de trem do qual depende e por último uma descrição do trem. Nesta figura, observa-se a descrição de um modelo de trem que depende do modelo de trem TO e cujo tempo de percurso em cada trecho será a metade do tempo associado ao modelo TO (fator de ajuste 0.5).

5.1.1.4. Atualização de dados: Modelos de percurso

A janela **Modelos de Percurso** (Figura 5.9) implementa a funcionalidade **Atualizar dados dos modelos de percurso**. Esta janela está dividida em dois painéis laterais. O painel da esquerda contém uma tabela com os dados dos modelos

de percurso enquanto que o painel da direita contém uma tabela com a descrição de um destes modelos. Quando o usuário seleciona um modelo de percurso da tabela da esquerda, os dados da tabela da direita são atualizados para refletir a descrição do modelo de percurso selecionado.

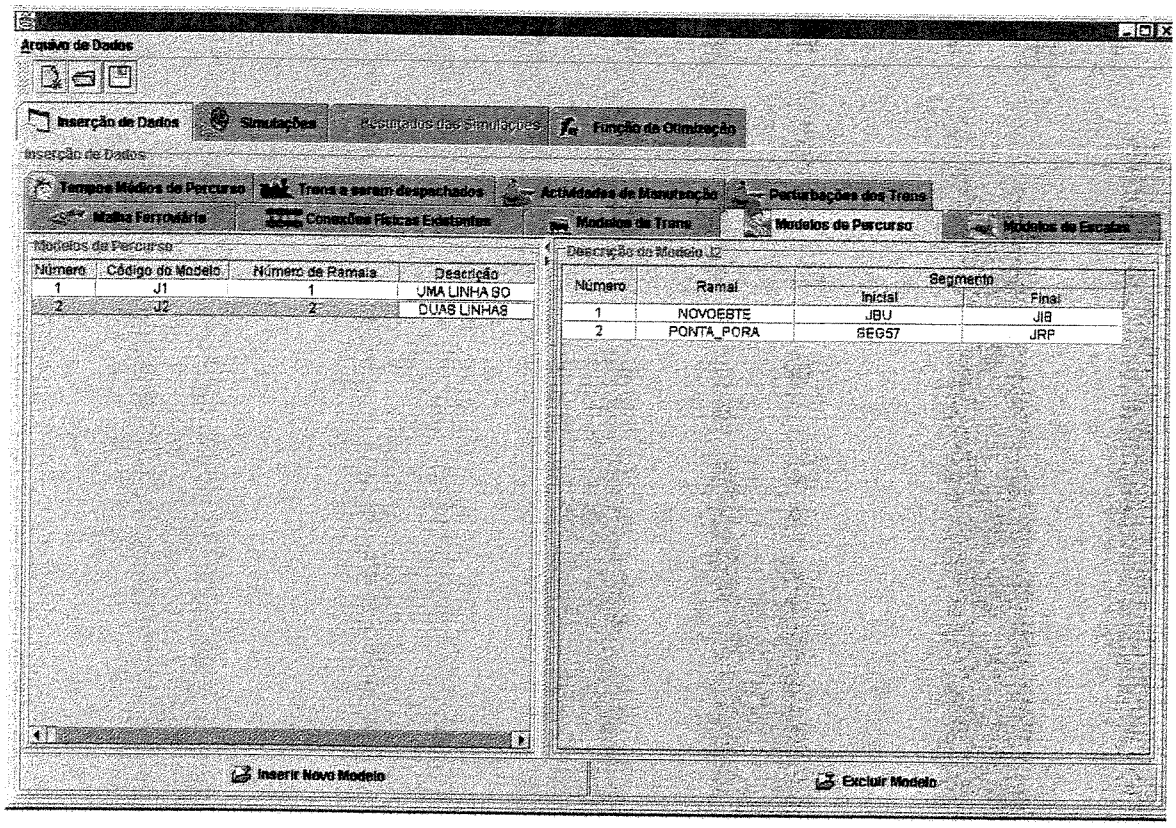


Figura 5.9 Janela: Modelos de percurso

Os dados dos modelos de percurso são um código; um número de ramais que percorre e uma descrição. Quando o usuário insere o número de ramais em um modelo de percurso, é criada – no painel do lado direito – uma tabela com o número de linhas necessárias para descrever o modelo de percurso. Um modelo de percurso é descrito indicando os segmentos inicial e final do(s) ramal(is) que percorre(m). No caso de percorrer mais de um ramal, supõe-se que o segmento final de um ramal está conectado com o segmento inicial de outro.

Na Figura 5.9 mostra-se a definição de dois modelos de percursos: J1 e J2. O primeiro deles corresponde ao ramal NOVOESTE como um todo. O modelo J2 contém dois ramais. Na tabela da direita observamos que este contém o ramal

NOVOESTE do segmento JBU até o segmento JIB, continuando até o ramal PONTA PORA do segmento SEG57 até o segmento JRP.

5.1.1.5. Atualização de dados: Modelos de escalas

A janela **Modelos de Escalas** (Figura 5.10) proporciona a funcionalidade **Atualizar dados dos modelos de escalas**. Esta janela está dividida em dois painéis laterais. O painel da esquerda contém uma tabela com os dados dos modelos de escalas enquanto que o painel da direita contém uma tabela com as paradas (escala) de um destes modelos. Quando o usuário seleciona um modelo de escala da tabela da esquerda, os dados da tabela da direita são atualizados para refletir a descrição do modelo de escala selecionado.

Os dados dos modelos de escalas são um código; um número de paradas do modelo e uma descrição. Quando o usuário insere o número de paradas, é criada – no painel do lado direito – uma tabela com o número de linhas necessárias para descrever cada parada. Cada parada é definida indicando o segmento do ramal onde será realizada, o tempo e o desvio que pode sofrer o tempo de parada.

Na Figura 5.10 ilustra três modelos de escalas: S1, S2 e S3. O modelo S1 realiza apenas uma parada no segmento JCB do ramal NOVOESTE. Esta parada dura três horas (03:00) e pode sofrer um desvio de vinte minutos (00:20).

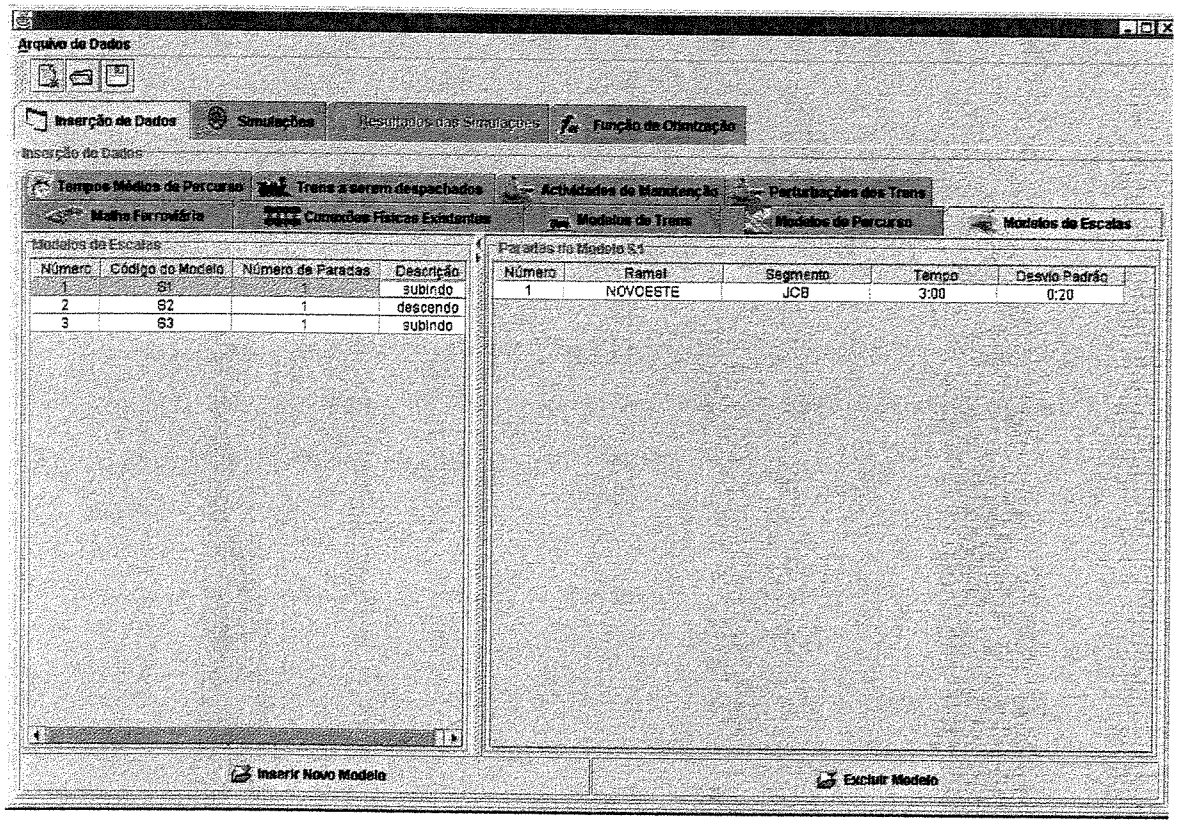
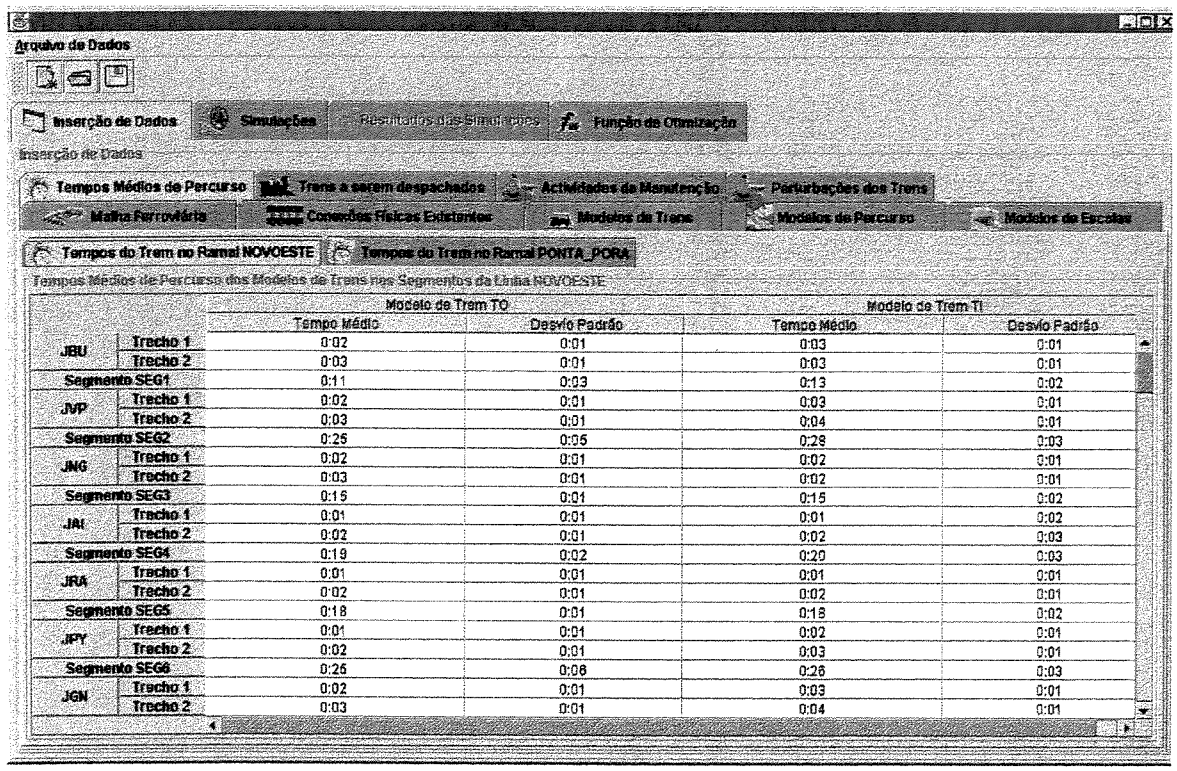


Figura 5.10 Janela: Modelos de escalas

5.1.1.6. Atualização de dados: Tempos médios de percurso

A janela **Tempos Médios de Percurso** (Figura 5.11) proporciona a funcionalidade **Atualizar tempos médios de percurso**. Esta janela contém um painel tabulado com dois painéis, um para cada ramal que compõe a malha ferroviária.

A Figura 5.11 mostra a janela **Tempos do Trem no Ramal NOVOESTE.** Esta janela contém uma tabela cujo lado esquerdo mostra os trechos dos segmentos do ramal NOVOESTE e na parte superior mostra os modelos de trens TO e TI. Para cada um dos modelos de trens define-se um tempo médio de percurso e um desvio.



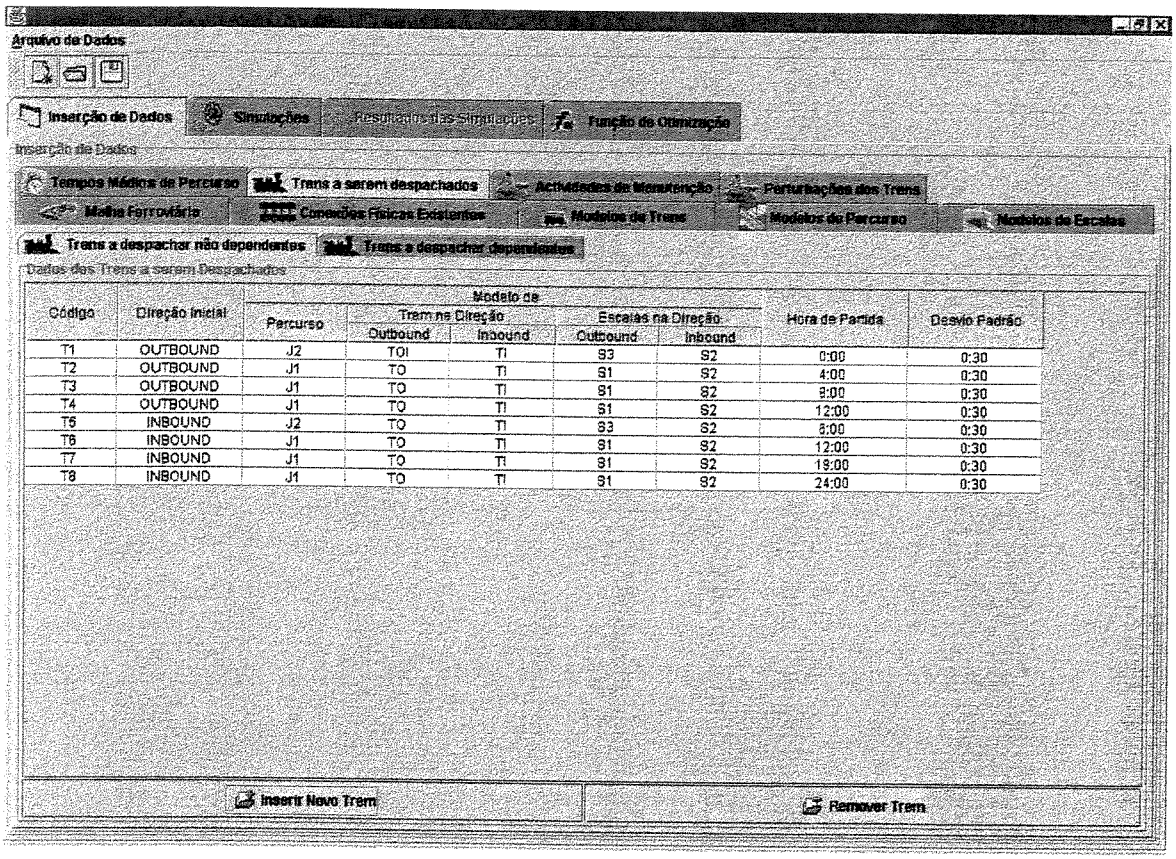
		Modelo de Trem TO		Modelo de Trem TI	
		Tempo Médio	Desvio Padrão	Tempo Médio	Desvio Padrão
JBU	Trecho 1	0:02	0:01	0:03	0:01
	Trecho 2	0:03	0:01	0:03	0:01
Segmento SEG1		0:11	0:03	0:13	0:02
JVP	Trecho 1	0:02	0:01	0:03	0:01
	Trecho 2	0:03	0:01	0:04	0:01
Segmento SEG2		0:25	0:05	0:29	0:03
JNG	Trecho 1	0:02	0:01	0:02	0:01
	Trecho 2	0:03	0:01	0:02	0:01
Segmento SEG3		0:15	0:01	0:15	0:02
JAI	Trecho 1	0:01	0:01	0:01	0:02
	Trecho 2	0:02	0:01	0:02	0:03
Segmento SEG4		0:19	0:02	0:20	0:03
JRA	Trecho 1	0:01	0:01	0:01	0:01
	Trecho 2	0:02	0:01	0:02	0:01
Segmento SEG5		0:18	0:01	0:18	0:02
JPY	Trecho 1	0:01	0:01	0:02	0:01
	Trecho 2	0:02	0:01	0:03	0:01
Segmento SEG6		0:26	0:06	0:26	0:03
JGN	Trecho 1	0:02	0:01	0:03	0:01
	Trecho 2	0:03	0:01	0:04	0:01

Figura 5.11 Janela: Tempos médios de percurso

5.1.1.7. Atualização de dados: Trens a serem despachados

A janela **Trens a serem despachados** (Figura 5.12) implementa a funcionalidade **Atualizar dados de trens a despachar**. Esta janela contém um painel tabulado com outros painéis. Um deles permite a atualização de trens cujos despachos não dependem de outros trens a despachar enquanto que o outro permite a atualização de trens cujos despachos dependem de outros trens.

A Figura 5.12 mostra a janela **Trens a Despachar não dependentes** a qual contém uma tabela que define os trens cujos despachos não dependem de outros trens. Os trens são definidos através de um código, um sentido inicial de viagem, um modelo de percurso, um modelo de trem no sentido *outbound* e outro no sentido *inbound*, um modelo de escalas no sentido *outbound* e outro no *inbound*, uma hora de partida e um desvio padrão desta hora de partida. Na Figura 5.12 definiu-se oito trens: T1, T2, T3, T4, T5, T6, T7 e T8, respectivamente.



Código	Direção Inicial	Percurso	Modelo de Trem na Direção		Escalas na Direção		Hora de Partida	Desvio Padrão
			Outbound	Inbound	Outbound	Inbound		
T1	OUTBOUND	J2	TOI	TI	S3	S2	0:00	0:30
T2	OUTBOUND	J1	TO	TI	S1	S2	4:00	0:30
T3	OUTBOUND	J1	TO	TI	S1	S2	8:00	0:30
T4	OUTBOUND	J1	TO	TI	S1	S2	12:00	0:30
T5	INBOUND	J2	TO	TI	S3	S2	8:00	0:30
T6	INBOUND	J1	TO	TI	S1	S2	12:00	0:30
T7	INBOUND	J1	TO	TI	S1	S2	19:00	0:30
T8	INBOUND	J1	TO	TI	S1	S2	24:00	0:30

Figura 5.12 Janela: Trens a despachar não dependentes

A Figura 5.13 mostra a janela **Trens a Despachar dependentes** com uma tabela que define os trens a serem despachados que são dependentes de outros trens. Estes trens são definidos por um código; um sentido inicial de viagem, um modelo de percurso, um modelo de trem no sentido *outbound* e outro no sentido *inbound*, um modelo de escalas no sentido *outbound* e outro no sentido *inbound*, um código do trem do qual depende, quanto tempo depois é despachado e um desvio deste tempo. Nesta figura definiu-se dois trens: T9 e T10, ambos dependentes do trem T1.

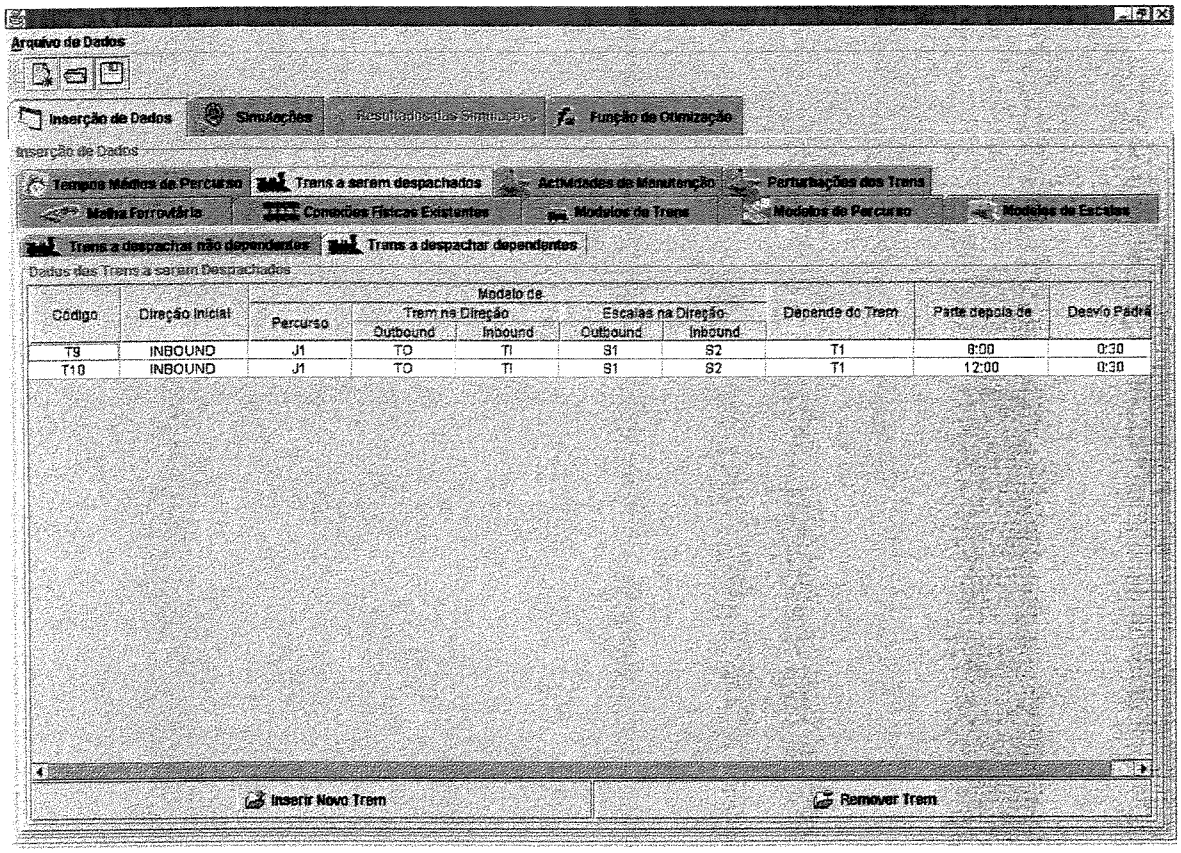


Figura 5.13 Janela: Trens a despachar dependentes

5.1.1.8. Atualização de dados: Atividades de manutenção

A janela **Atividades de Manutenção** (Figura 5.14) implementa a funcionalidade **Atualizar atividades de manutenção**. Esta janela contém uma tabela na qual definem-se as atividades de manutenção. Cada atividade está definida por um código; um trecho de um segmento de um ramal; uma hora de início e o desvio desta hora; uma duração e o desvio desta duração. No exemplo da Figura 5.14 define-se a atividade A1. Esta atividade será realizada no trecho 1 do segmento SEG16 do ramal NOVOESTE, começa as dez horas (10:00) com um desvio de uma hora (01:00) e prolonga-se por quatro horas (04:00) podendo sofrer um desvio de uma hora (01:00).

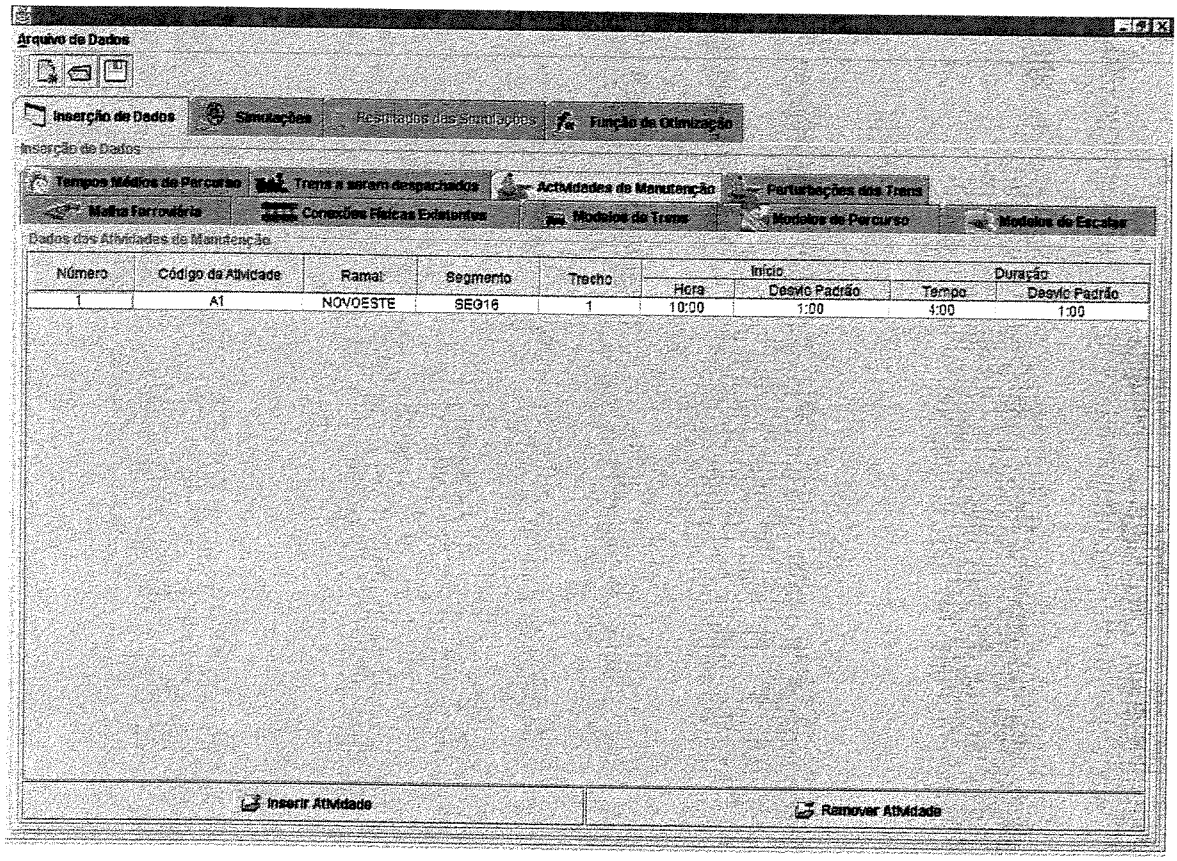


Figura 5.14 Janela: Atividades de manutenção

5.1.1.9. Atualização de dados: Perturbações dos trens

A janela **Pertubações dos Trens** (Figura 5.15) implementa a funcionalidade **Atualizar dados de perturbações que podem sofrer os trens**. Esta janela contém uma tabela na qual definem-se as perturbações. Cada perturbação está definida por o código do trem que sofrerá a perturbação; uma hora de início da perturbação e o desvio desta hora; uma duração e o desvio desta duração e uma probabilidade de ocorrência da perturbação. No exemplo da Figura 5.15 define-se uma perturbação para o trem T2, a probabilidade de ocorrência desta perturbação é de 0.5, começa as doze horas (12:00) com um desvio de uma hora (01:00) e prolonga-se por meia hora (00:30) podendo sofrer um desvio de quinze minutos (00:15).

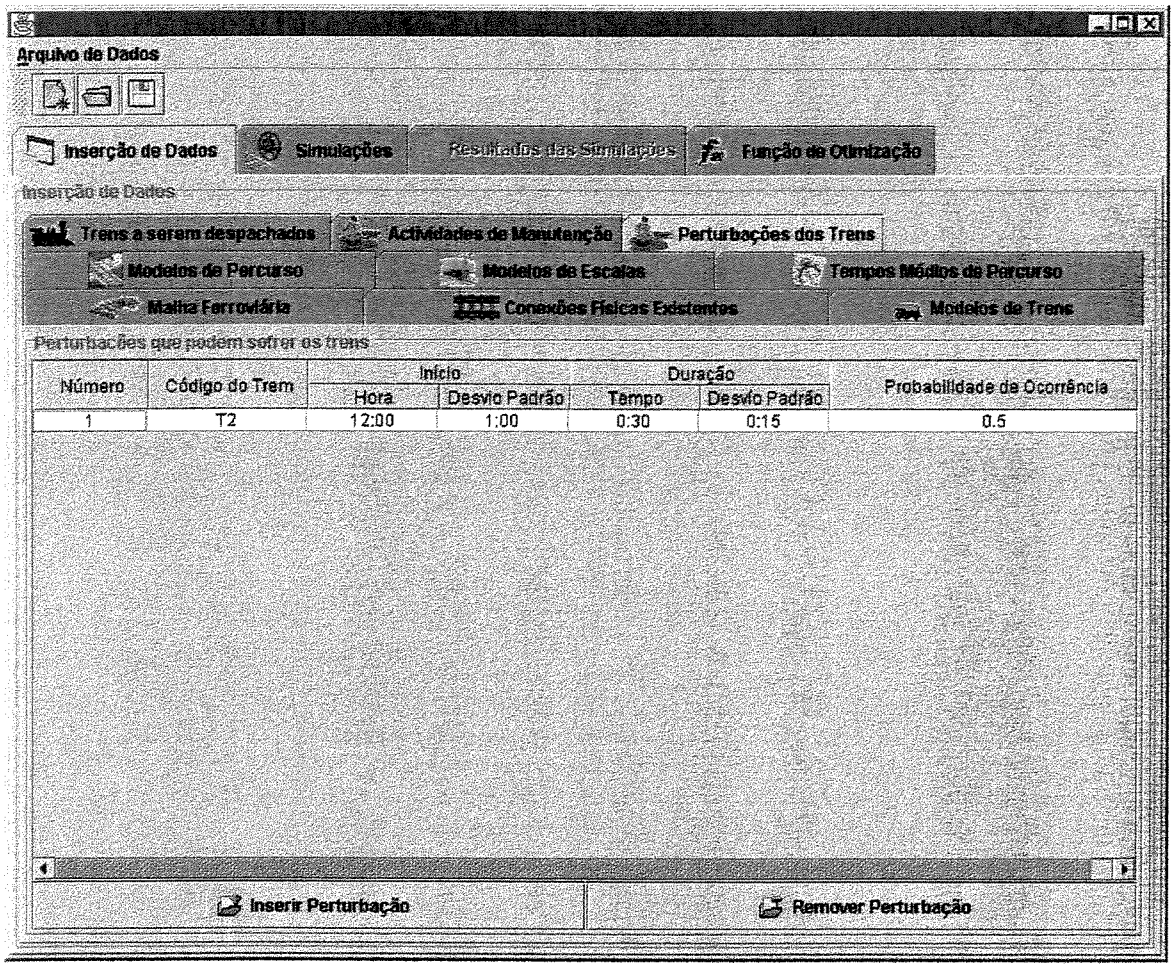


Figura 5.15 Janela: Perturbações dos trens

5.1.2. Interface com o usuário: Despachar trens

Esta seção descreve como o sistema implementa a funcionalidade de **Despachar trens**, a qual é fornecida no painel **Simulações**. Como foi mencionado no capítulo anterior, esta funcionalidade tem como objetivo permitir ao usuário configurar a(s) simulação(ões) a serem realizadas e dar início à(s) mesma(s). A Figura 5.16 mostra a janela que implementa esta funcionalidade.

A parte superior da janela da Figura 5.16 apresenta a configuração de parâmetros da simulação. Estes parâmetros são: número de dias simulados, o número de simulações a realizar, o nome de arquivo de dados da simulação, o nome do arquivo da função de otimização e o uso de dados determinísticos. Se a simulação é realizada com dados determinísticos, os desvios definidos para alguns dos dados são ignorados. O centro da

janela permite ao usuário escolher os resultados que deseja obter. Escolhem-se os resultados desejados de cada simulação realizada e os resultados que reúnem a todas as simulações realizadas. Estes resultados são explicados na seção seguinte. Na parte inferior da janela existe um botão com o qual o usuário dá início à(s) simulação(es) e uma barra que mostra o progresso no tempo das simulações.

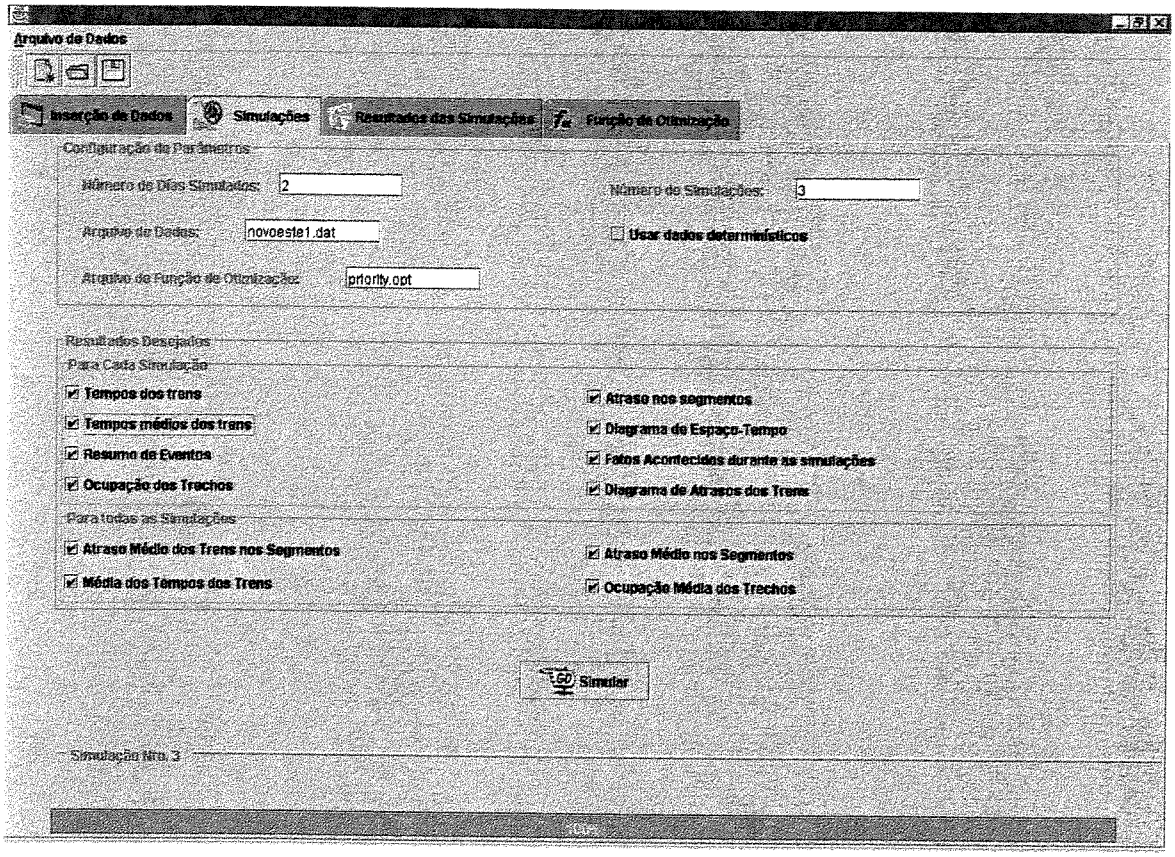


Figura 5.16 Janela: Despachar trens

A Figura 5.16 mostra que foram realizadas três simulações, todas elas com um horizonte de dois dias. Estas simulações estão baseadas nos dados do modelo armazenados no arquivo "novoeste1.dat" (arquivo que contém os dados que vem sendo usados como exemplo). A função de otimização usada nas simulações está descrita no arquivo "priority.opt".

5.1.3. Interface com o usuário: Reportar informação

Esta seção descreve como o sistema implementa a funcionalidade de **Reportar Informação**, a qual está relacionada com o painel de **Resultados das Simulações**.

Como foi mencionado no capítulo anterior, esta funcionalidade tem como objetivo permitir ao usuário observar os diferentes resultados obtidos e armazená-los. A Figura 5.17 mostra a janela que implementa esta funcionalidade. Esta janela contém um painel tabulado que contém outros quatro painéis. Os três primeiros apresentam os resultados obtidos em três simulações e o último apresenta os resultados obtidos para todas as simulações. A seguir são descritos estes painéis.

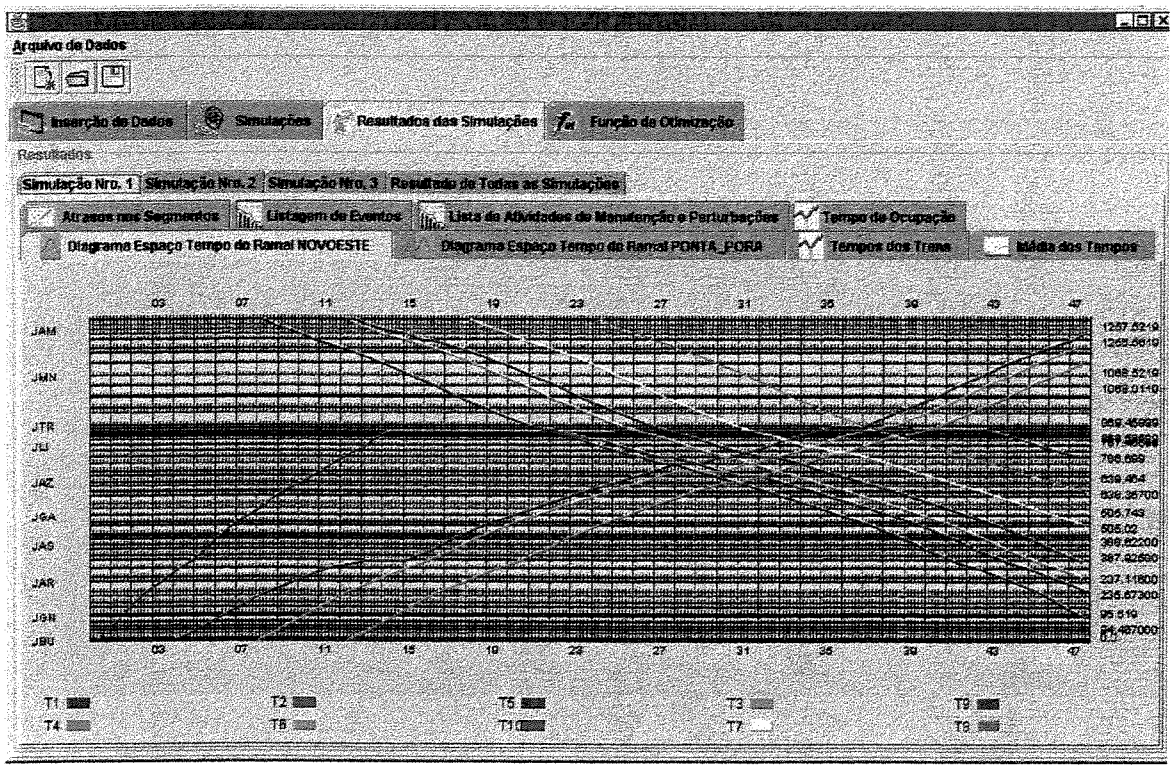


Figura 5.17 Janela: Resultados para cada simulação

5.1.3.1. Resultados obtidos para cada simulação

A janela **Simulação Nro. 1** da Figura 5.17 apresenta os resultados obtidos para a primeira simulação realizada. Esta janela apresenta um painel tabulado que contém painéis que mostram os diferentes resultados obtidos para esta simulação. O horizonte de simulação é 2 dias e despacha dez trens na malha ferroviária que vem sendo usada como exemplo. A seguir explica-se cada um dos resultados.

O primeiro resultado obtido é o diagrama espaço tempo. Este diagrama pode ser observado na Figura 5.17. Este mostra graficamente o deslocamento dos trens na linha ferroviária durante o horizonte de simulação. O eixo horizontal representa o

horizonte de simulação (2 dias = 48 horas) e o eixo vertical representa a linha ferroviária. Dado que na simulação realizada foi considerado o deslocamento de trens nas linhas NOVOESTE e PONTA PORÃ, são apresentados dois diagramas espaço tempo, cada um dos quais referido a uma das linhas. Quando o horizonte de simulação é muito grande e o comprimento da linha é muito maior, é difícil enxergar o deslocamentos dos trens em um diagrama só, sendo preferível dividir o diagrama geral em pequenos diagramas ou fazer crescer uma área de interesse. Basta com que o usuário selecione uma área do diagrama com o mouse para obter um sub-diagrama espaço tempo da área selecionada. A Figura 5.18 apresenta um sub-diagrama espaço tempo que mostra com mais detalhe o estado da linha NOVOESTE, entre os segmentos JAR e JAU, no período das doze horas (12:00) as dezessete horas (17:00). Nesta figura, vemos que o trem T2 (linha vermelha) é atrasado no segmento JMP até as quinze horas e nove minutos (15:09) porque no segmento seguinte SEG16 está sendo realizada uma atividade de manutenção (veja Figura 5.23).

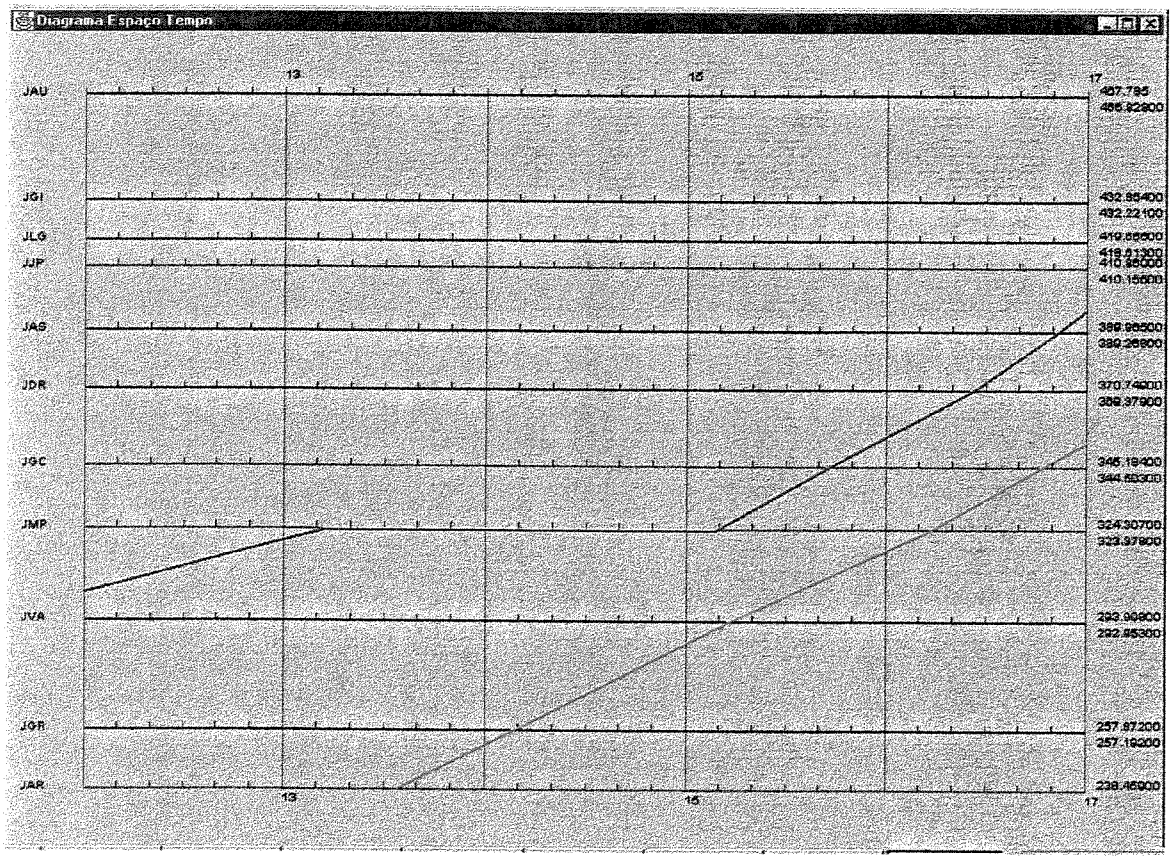


Figura 5.18 Janela: Diagrama de espaço tempo

Um outro resultado obtido são os **Tempos dos Trens**, a Figura 5.19 apresenta estes resultados. Esta janela contém um painel tabulado o qual contém painéis para reportar os tempos de cada um dos trens despachados durante a simulação.

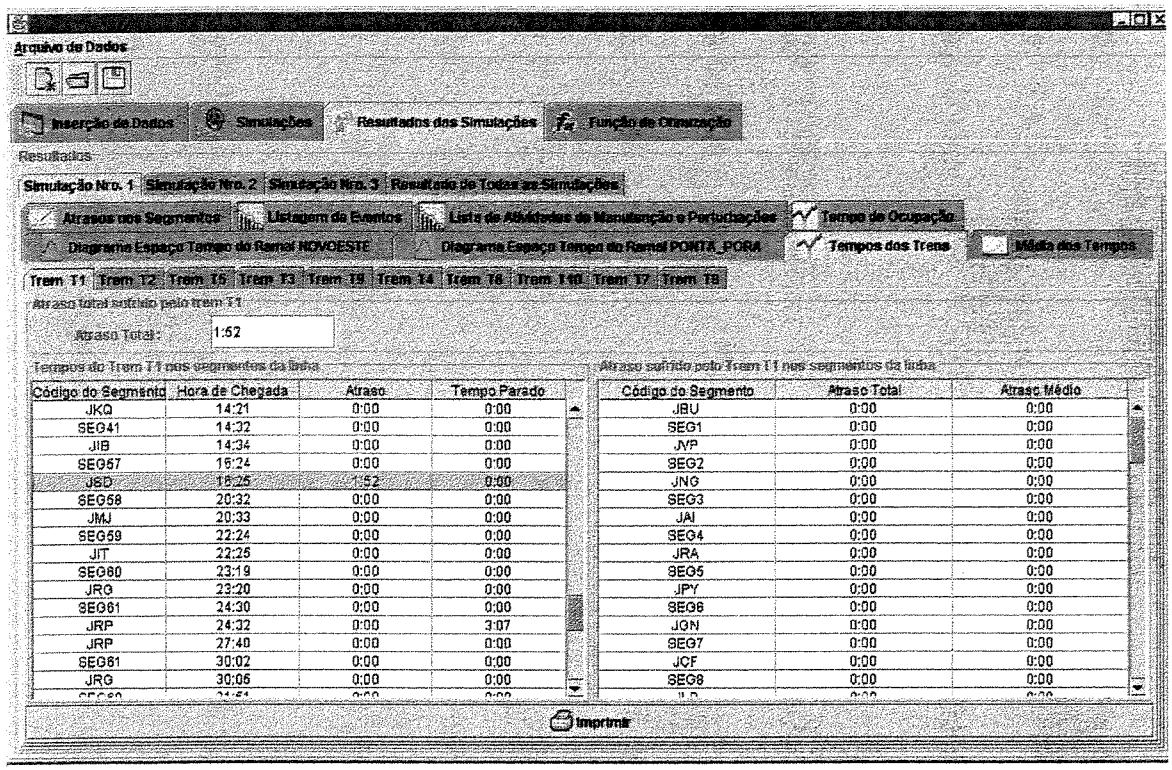


Figura 5.19 Janela: Tempos dos trens

A Figura 5.19 mostra os resultados obtidos para o trem T1. O atraso total sofrido pelo trem foi uma hora e cinqüenta e dois minutos (01:52). A tabela da esquerda mostra os tempos do trem T1 nos segmentos: a que horas chegou, quanto tempo ficou atrasado e quanto tempo fico parado fazendo operações (e.g. carregando ou descarregando). A tabela da direita mostra o atraso total e médio sofrido pelo trem T1 nos diferentes segmentos. O atraso médio é resultado do atraso total dividido entre o número de vezes que ficou atrasado. Nesta figura, observa-se que o atraso do trem aconteceu no segmento JSD. Na parte inferior da janela aparece o botão **Imprimir**, o qual permite imprimir a tabela selecionada.

O sistema fornece os resultados **Média dos Tempos**, apresentados na Figura 5.20. Esta figura mostra quatro tabelas. A primeira tabela apresenta a média dos tempos quando os trens viajam no sentido *outbound*. Ela apresenta qual foi o atraso

médio sofrido por cada um dos trens despachados quando viajavam no sentido *outbound* e quanto tempo em média demoraram para completar o seu percurso neste sentido. Na figura, observa-se que o trem T1 sofreu um atraso médio de uma hora e cinqüenta e dois minutos (1:52) e levou em média vinte e quatro horas e catorze minutos (24:14) para percorrer seu percurso no sentido *outbound*. A tabela do lado, apresenta a mesma informação mas no sentido *inbound*. Na figura, observa-se que o trem T1 não sofreu atraso quando viajava no sentido *inbound*. Este trem, nos dois dias de simulação, não completou sua viagem neste sentido, é por isso que o tempo médio que levou para percorrer seu trajeto neste sentido aparece como zero horas (00:00). A terceira tabela, mostra a média dos tempos nos segmentos terminais. Esta tabela apresenta o atraso médio sofrido por cada um dos trens nos segmentos terminais e o tempo médio que estes ficaram fazendo operações neles. A última tabela mostra os tempos médios nos ciclos de viagem. Um trem completa um ciclo de viagem quando volta ao ponto de onde partiu originalmente. No exemplo apresentado, nenhum trem completou um ciclo de viagem.

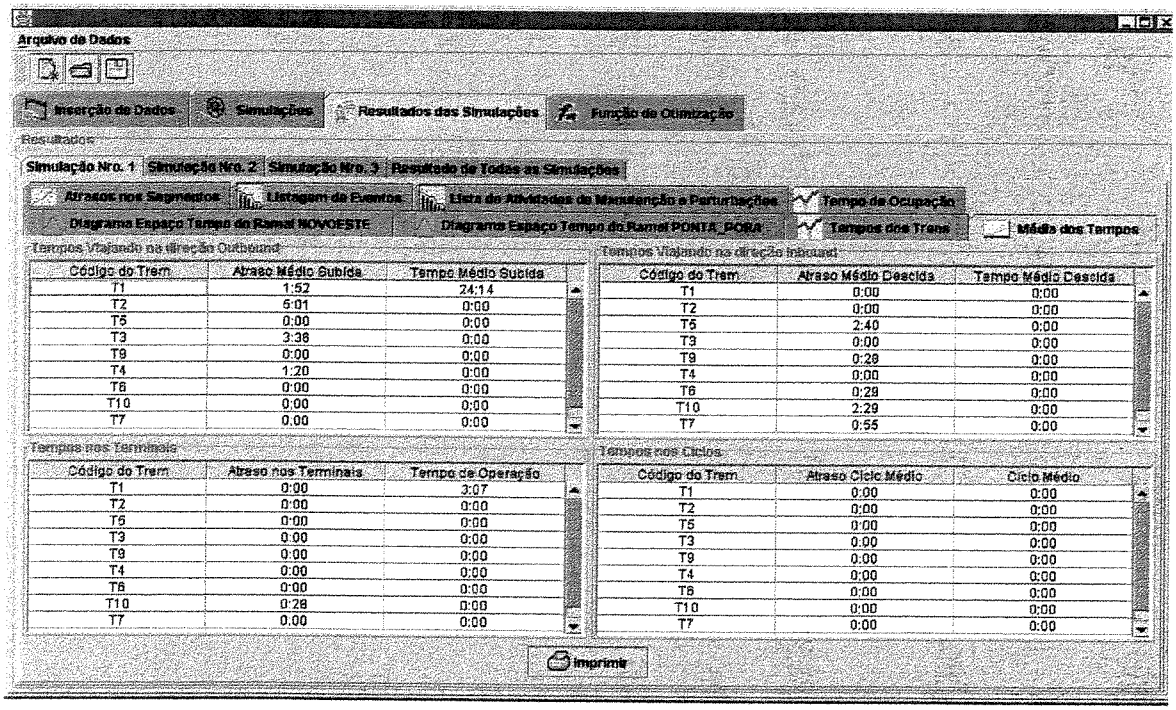


Figura 5.20 Janela: Média dos tempos

O sistema também fornece ao usuário o atraso sofrido em cada segmento da malha. Este resultado é mostrado na janela **Atrasos nos Segmentos** apresentada

na Figura 5.21. Nesta figura, observa-se que no segmento JMP o atraso total sofrido foi de uma hora e cinqüenta e seis minutos (1:56).

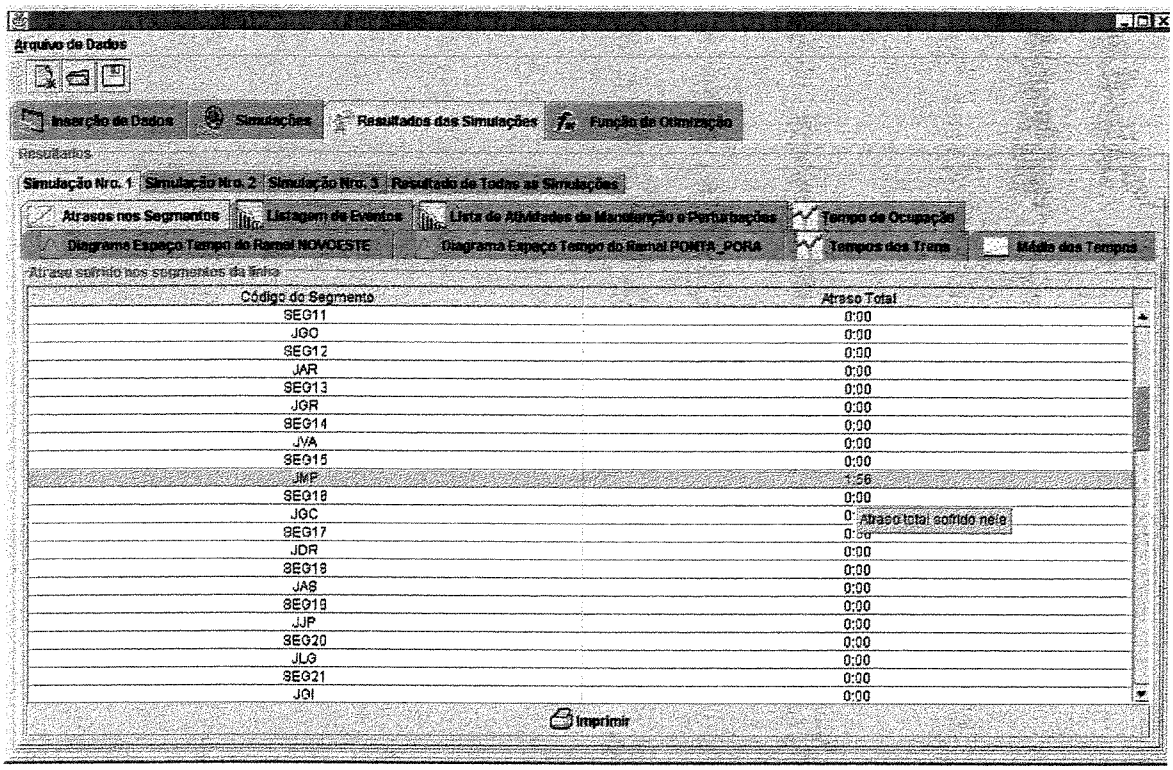
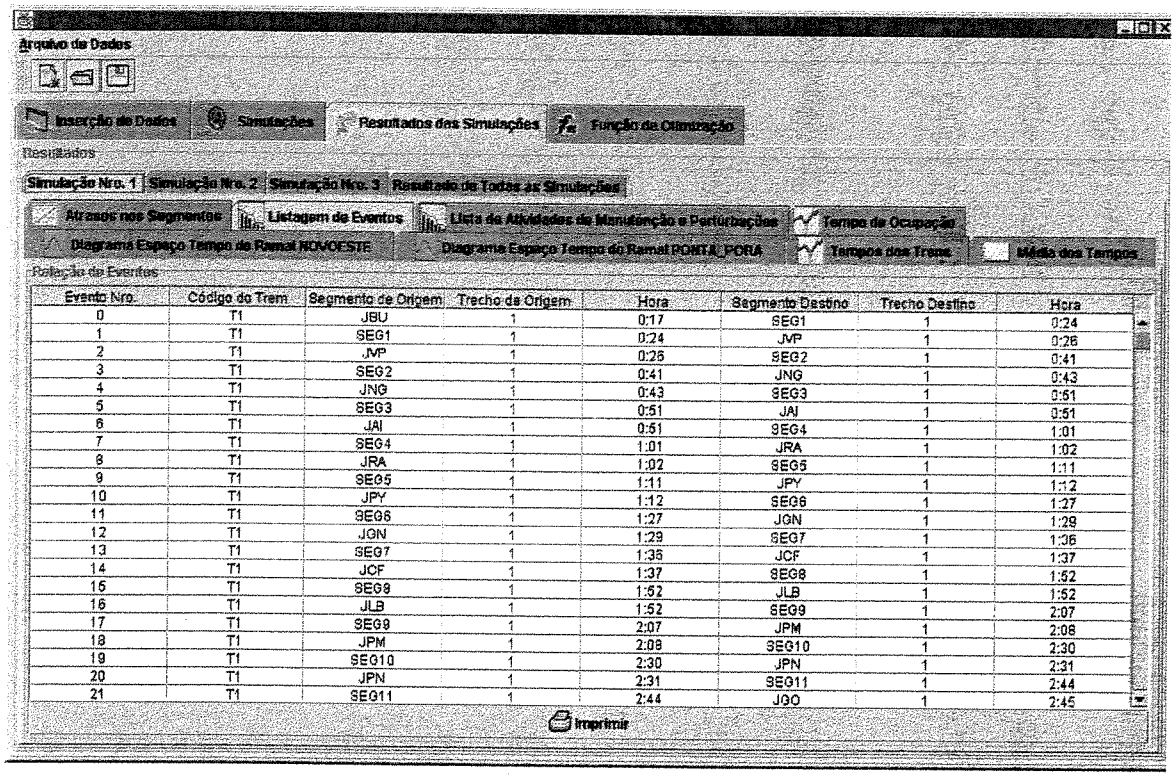


Figura 5.21 Janela: Atrasos nos segmentos

Um outro resultado fornecido é a **Listagem de Eventos** a qual é apresentada na Figura 5.22. Esta figura mostra uma tabela descrevendo cada um dos eventos acontecidos no sistema referidos a trens sendo despachados. Na figura observamos que o trem T1 saiu de JBU às zero horas e dezessete minutos (00:17) e atravessou o segmento SEG1 às zero horas e vinte e quatro minutos (00:24), em seguida levou dois minutos (00:02) para atravessar o segmento JVP, às zero horas e quarenta e um minutos (00:41) atravessou o SEG2 e assim sucessivamente.

O sistema apresenta ao usuário a lista de atividades de manutenção e perturbações que sofreram os trens durante a simulação. Este resultado é mostrado na janela **Lista de Atividades de Manutenção e Perturbações,** apresentada na Figura 5.23. Nesta figura observamos duas tabelas. A primeira delas apresenta onde e quando foram realizadas as atividades de manutenção. A segunda, mostra quando trens foram perturbados. Segundo a figura, foi dada manutenção ao segmento SEG16 das dez horas e vinte e quatro minutos (10:24) até às quinze horas e sete

minutos (15:07) e o trem T2 foi perturbado das doze horas e vinte e seis minutos (12:26) até às treze horas e dez minutos (13:10).



Evento Nro.	Código do Trem	Segmento de Origem	Trecho da Origem	Hora	Segmento Destino	Trecho Destino	Hora
0	T1	JBU	1	0:17	SEG1	1	0:24
1	T1	SEG1	1	0:24	JVP	1	0:26
2	T1	JVP	1	0:26	SEG2	1	0:41
3	T1	SEG2	1	0:41	JNG	1	0:43
4	T1	JNG	1	0:43	SEG3	1	0:51
5	T1	SEG3	1	0:51	JAI	1	0:51
6	T1	JAI	1	0:51	SEG4	1	1:01
7	T1	SEG4	1	1:01	JRA	1	1:02
8	T1	JRA	1	1:02	SEG5	1	1:11
9	T1	SEG5	1	1:11	JPY	1	1:12
10	T1	JPY	1	1:12	SEG6	1	1:27
11	T1	SEG6	1	1:27	JGN	1	1:29
12	T1	JGN	1	1:29	SEG7	1	1:36
13	T1	SEG7	1	1:36	JCF	1	1:37
14	T1	JCF	1	1:37	SEG8	1	1:52
15	T1	SEG8	1	1:52	JLB	1	1:52
16	T1	JLB	1	1:52	SEG9	1	2:07
17	T1	SEG9	1	2:07	JPM	1	2:08
18	T1	JPM	1	2:08	SEG10	1	2:30
19	T1	SEG10	1	2:30	JPN	1	2:31
20	T1	JPN	1	2:31	SEG11	1	2:44
21	T1	SEG11	1	2:44	JGO	1	2:45

Figura 5.22 Janela: Listagem de eventos

Um outro resultado obtido é os **Tempos de Ocupação** o qual é apresentado na Figura 5.24. Nesta figura observamos uma tabela contendo todos os trechos da malha ferroviária e apresentando o tempo que cada um dos trechos foi ocupado, a média deste tempo e quanto tempo os trens ficaram estacionados nos trechos.

E por último, o sistema fornece um diagrama de barras mostrando o atraso total sofrido pelos trens. Este resultado é mostrado na janela **Diagrama de Atrasos dos Trens** apresentada na Figura 5.25. Vemos por exemplo que o trem T2 foi aquele que sofreu mais atraso cinco horas (05:00).

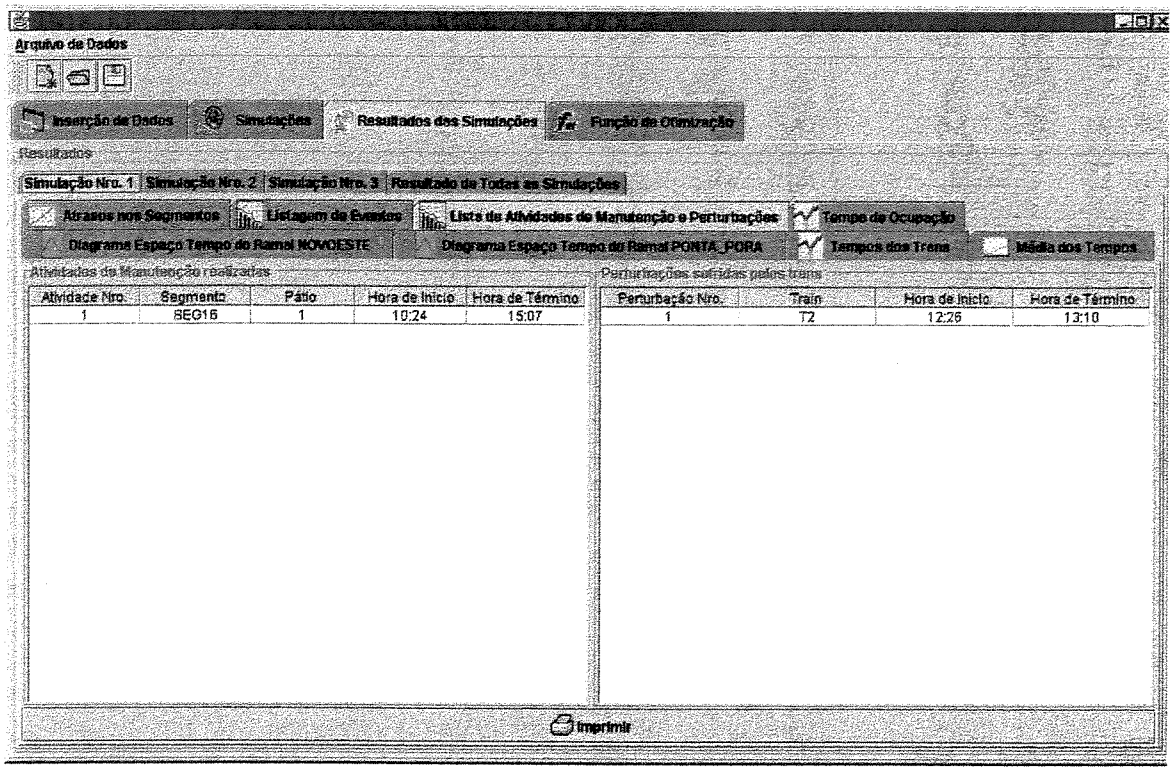


Figura 5.23 Janela: Lista de atividades de manutenção e perturbações

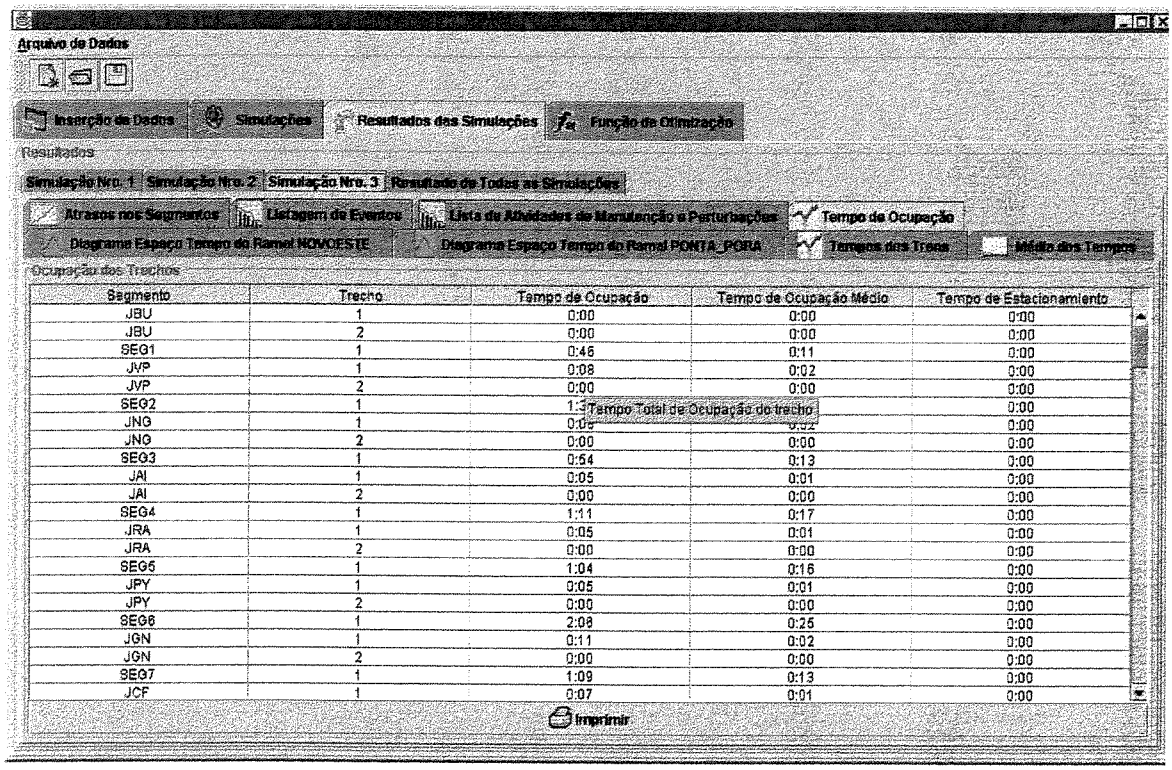


Figura 5.24 Janela: Tempo de ocupação

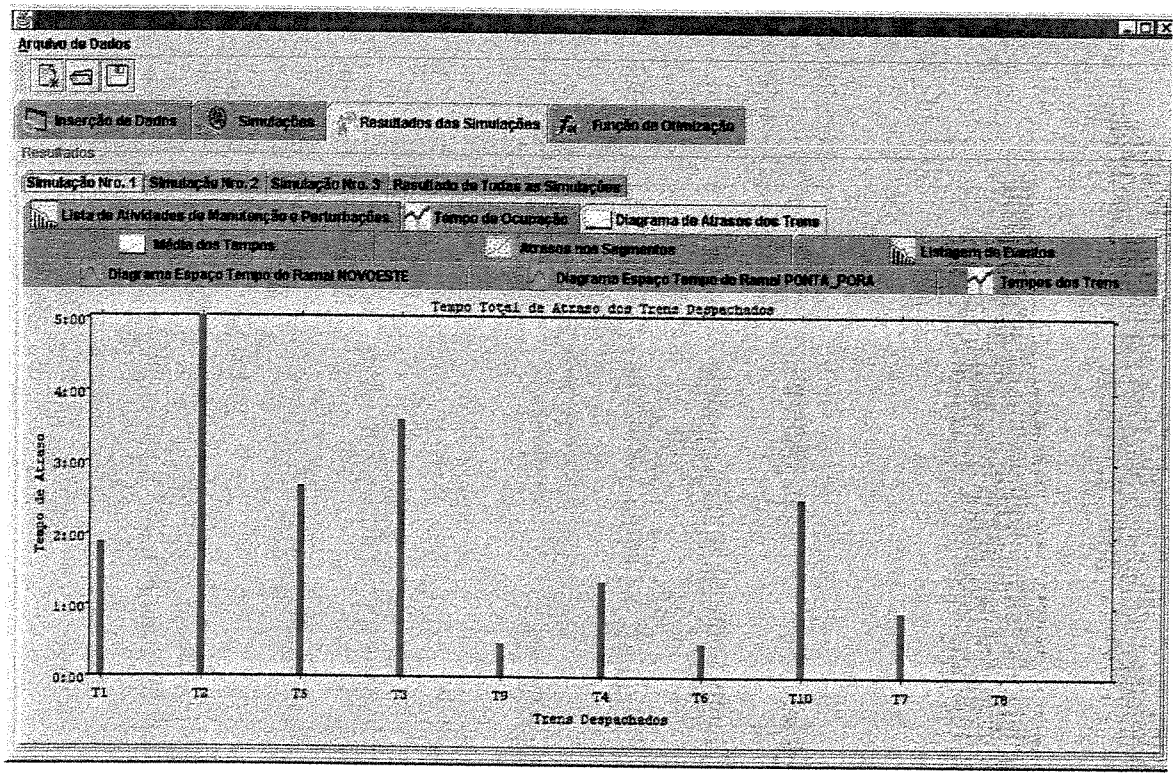


Figura 5.25 Janela: Diagrama de atrasos dos trens

5.1.3.2. Resultados obtidos para todas as simulações

O sistema fornece resultados baseados em dados de todas as simulações realizadas. Estes resultados foram criados por razões estatísticas e representam a média dos valores de todas as simulações. O sistema permite obter quatro resultados deste tipo.

A Figura 5.26 mostra a janela dos **Atrasos dos trens nos segmentos**. Esta janela contém uma tabela que apresenta a média do atraso total e a média do atraso médio (veja Figura 5.19) sofrido por todos os trens durante todas as simulações. Na parte superior da janela, mostra-se a somatória do atraso total sofrido por todos os trens e a somatória do atraso médio sofrido por eles (vinte e sete horas e quarenta e cinco minutos (27:45)).

A Figura 5.27 mostra a média de todas as simulações referidos aos dados apresentados na Figura 5.20. A Figura 5.28 mostra a média de todas as simulações

referidos aos dados apresentados na Figura 5.21 e por último, a Figura 5.29 mostra a média de todas as simulações referidos aos dados apresentados na Figura 5.24.

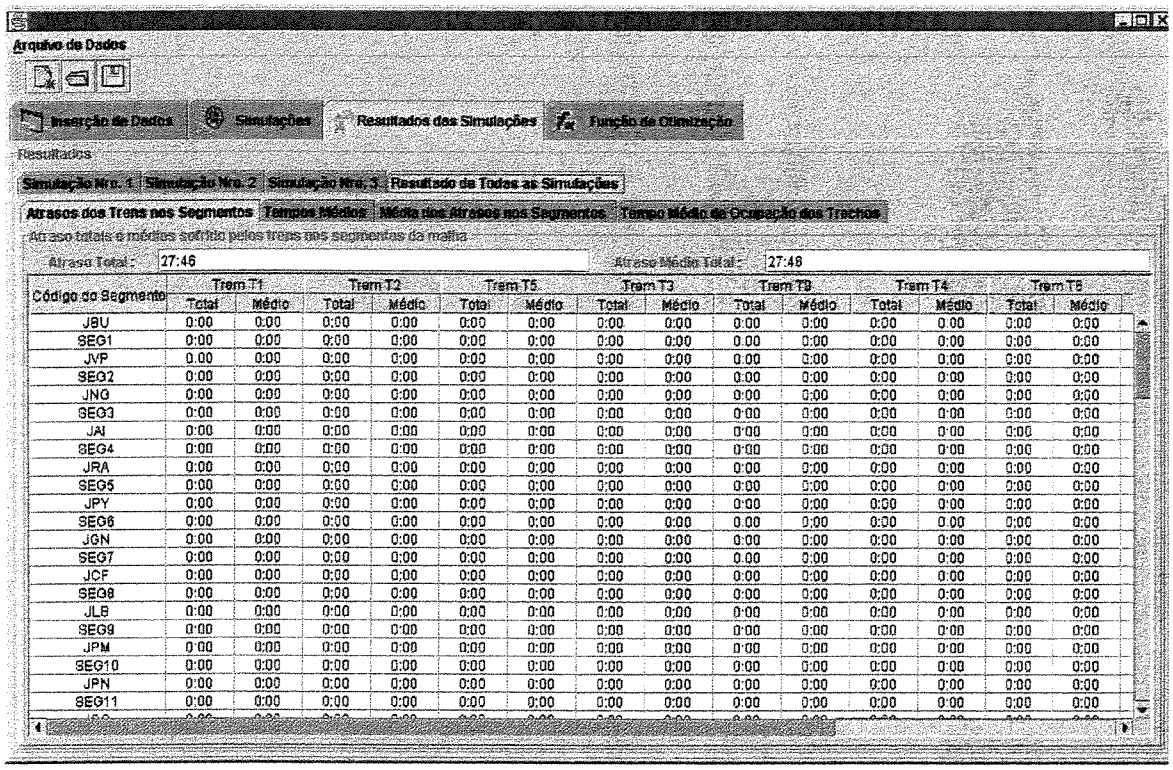


Figura 5.26 Janela: Atraso dos trens nos segmentos

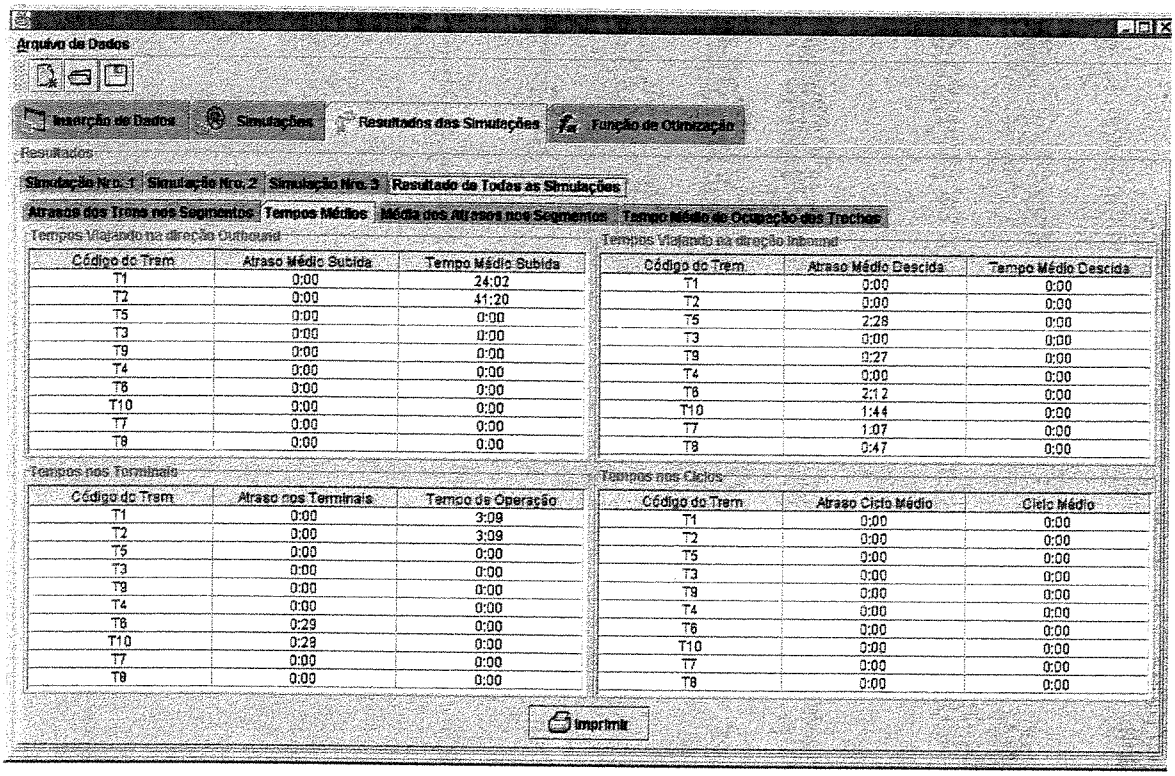


Figura 5.27 Janela: Tempos médios

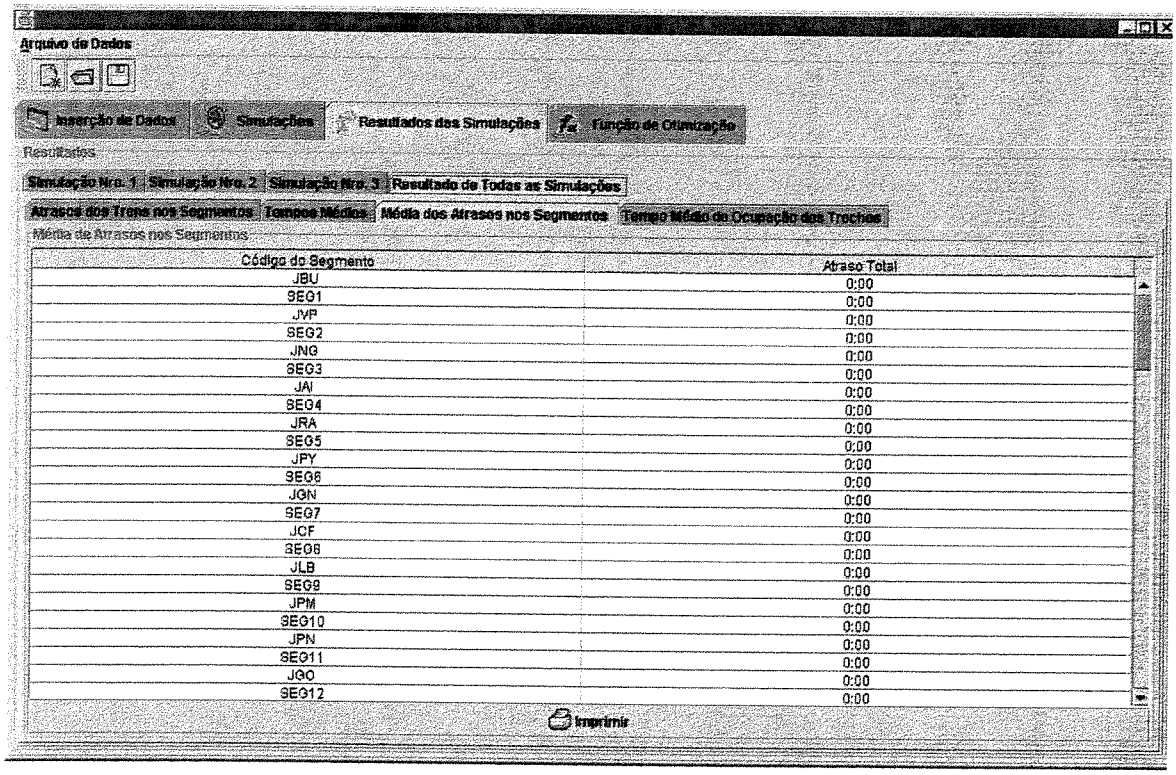
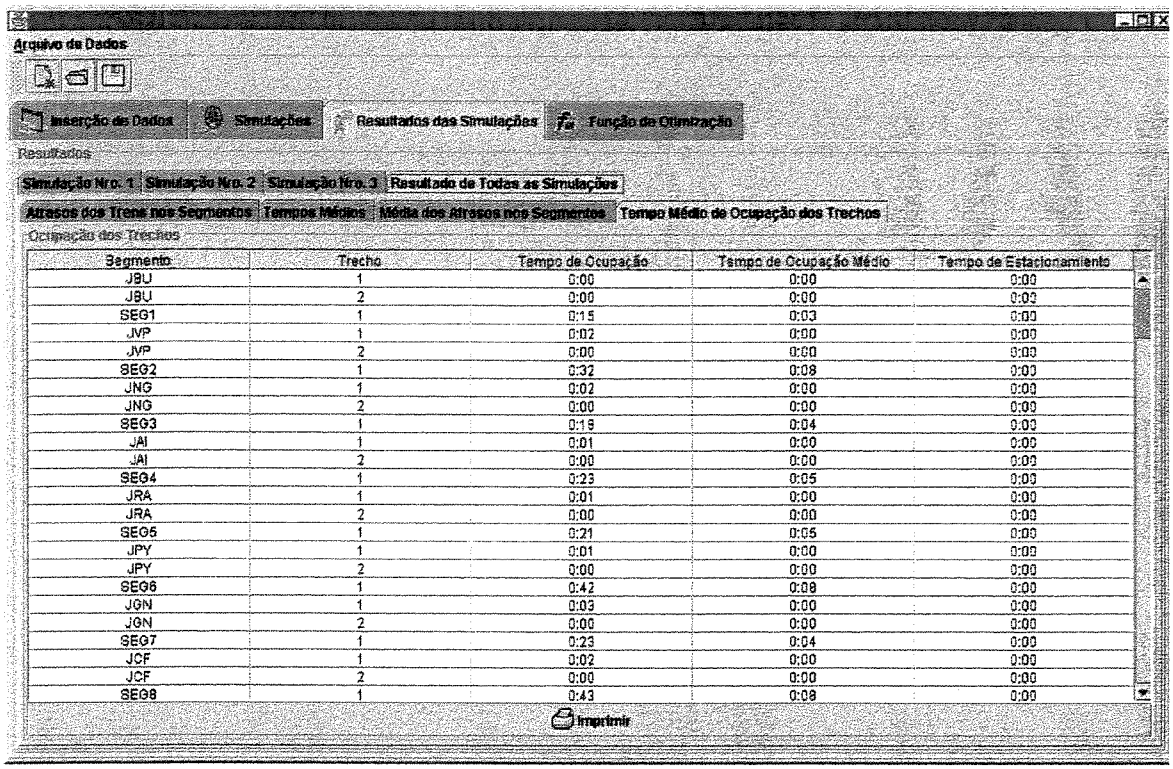


Figura 5.28 Janela: Média dos atrasos nos segmentos



Segmento	Trecho	Tempo de Ocupação	Tempo de Ocupação Médio	Tempo de Estacionamiento
JBU	1	0:00	0:00	0:00
JBU	2	0:00	0:00	0:00
SEG1	1	0:15	0:03	0:00
JVP	1	0:02	0:00	0:00
JVP	2	0:00	0:00	0:00
SEG2	1	0:32	0:08	0:00
JNG	1	0:02	0:00	0:00
JNG	2	0:00	0:00	0:00
SEG3	1	0:19	0:04	0:00
JAI	1	0:01	0:00	0:00
JAI	2	0:00	0:00	0:00
SEG4	1	0:23	0:05	0:00
JRA	1	0:01	0:00	0:00
JRA	2	0:00	0:00	0:00
SEG5	1	0:21	0:05	0:00
JPY	1	0:01	0:00	0:00
JPY	2	0:00	0:00	0:00
SEG6	1	0:42	0:08	0:00
JGN	1	0:03	0:00	0:00
JGN	2	0:00	0:00	0:00
SEG7	1	0:23	0:04	0:00
JCF	1	0:02	0:00	0:00
JCF	2	0:00	0:00	0:00
SEG8	1	0:43	0:08	0:00

Figura 5.29 Janela: Tempo médio de ocupação dos trechos

Estes são todos os resultados fornecidos pelo sistema computacional implementado. Usando estes resultados os usuários são capazes de realizar análises da performance dos trens, da capacidade da linha, preparar e avaliar os horários de saída dos trens, calcular o impacto de atividades de manutenção, avaliar funções de otimização, analisar diferentes configurações da malha ferroviária, etc. Na seção 5.2 apresentam-se algumas das possíveis análises que podem ser realizadas a partir dos resultados obtidos com o sistema.

5.1.4. Interface com o usuário: Atualizar funções de otimização

Esta seção descreve como o sistema implementa a funcionalidade de **Atualizar Funções de Otimização**, a qual está relacionada com o painel **Função de Otimização**. Como foi mencionado no capítulo anterior, esta funcionalidade tem como objetivo permitir ao usuário definir funções de otimização de despacho, armazená-las em arquivos de texto e se for o caso modificar o conteúdo destes arquivos. A Figura 5.30 mostra a implementação desta funcionalidade.

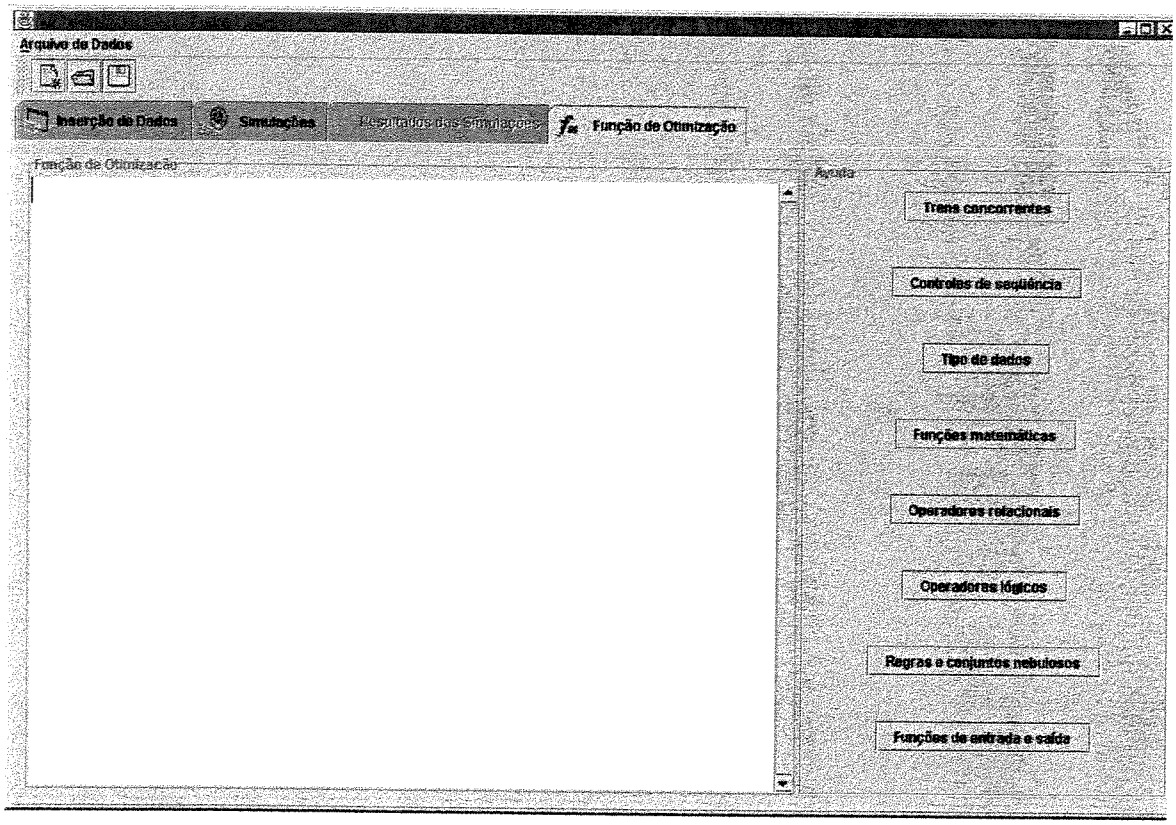


Figura 5.30 Janela: Função de otimização

Na Figura 5.30 observamos que a janela **Função de Otimização** consta de dois painéis: um painel para permitir editar funções de otimização de despacho e outro para fornecer ajuda durante a definição destas funções. Esta ajuda está relacionada com os componentes que podem ser usados para definir funções, ou seja: informação relacionada com os trens concorrentes, controles de seqüência (for, if, etc.), tipos de dados (int, double, etc.), funções matemáticas (sin, cos, etc.), operadores relacionais (<, <=, etc.), operadores lógicos (&&, ||, etc.), regras e conjuntos nebulosos (trapese, triang, etc.) e funções de entrada e saída de dados (read e write). Na Figura 5.31 observa-se a tabela de ajuda que é apresentada ao usuário quando este pressiona no botão **Trens concorrentes**. Fazendo uso da barra de ferramentas e do menu **Arquivo de Dados** o usuário pode abrir e salvar arquivos de funções de otimização. Estes arquivos são de extensão "opt".

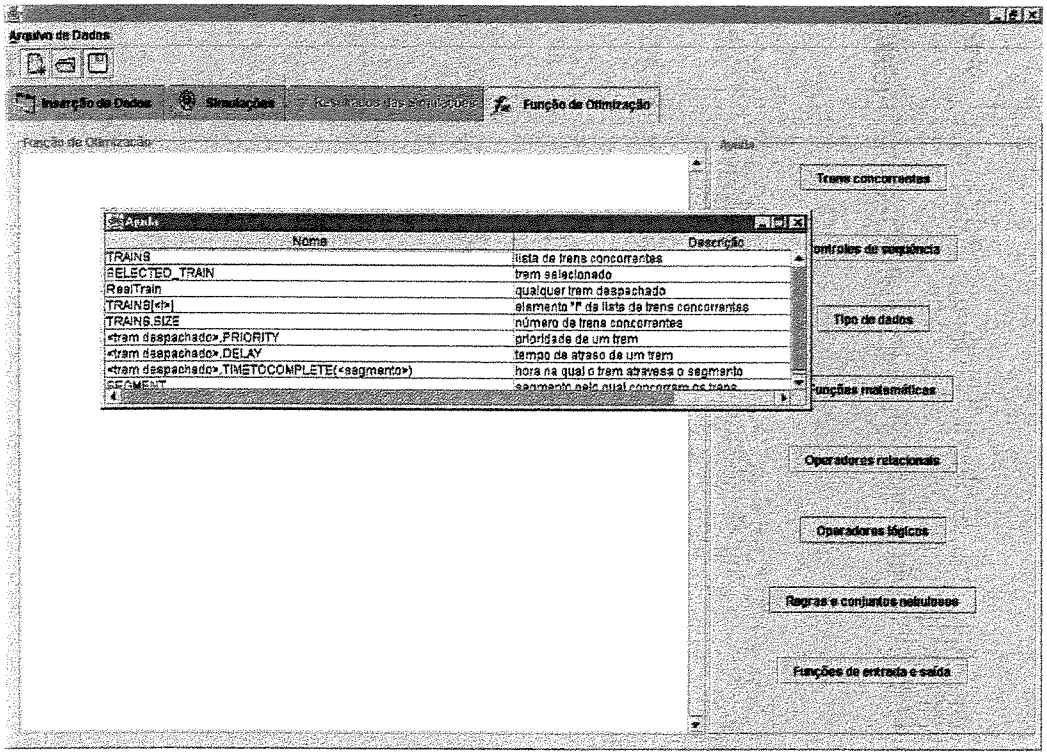


Figura 5.31 Janela: Ajuda na atualização de funções de otimização

5.2. Exemplos de Aplicação

Nesta seção apresenta-se exemplos de análises realizados. Estes exemplos foram construídos a partir de uma linha imaginária de 435,854 km de comprimento. Esta linha foi inspirada em dados fornecidos pela NOVOESTE. A Figura 5.32 mostra a linha com diferentes números de pátios intermediários (segmentos com 2 trechos paralelos). Todas as simulações foram realizadas para um horizonte de dois dias. Os resultados apresentados em cada exemplo são a média de dez simulações realizadas para cada um deles.

5.2.1. Exemplo: Avaliar configuração da linha

Este exemplo visa apresentar o uso do sistema na avaliação da configuração de uma linha ferroviária.

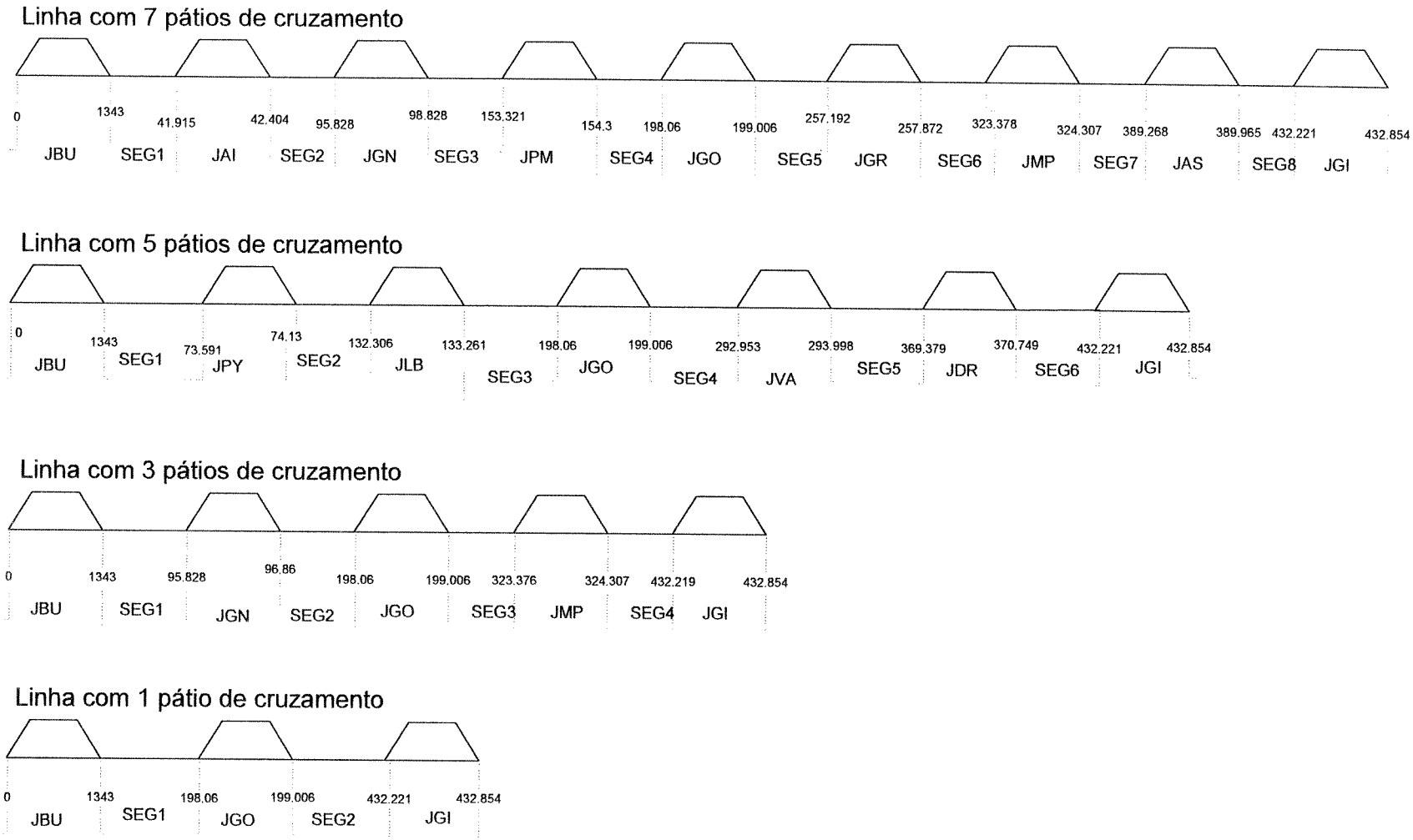


Figura 5.32 Exemplo de linhas ferroviárias

Conforme dito, o exemplo supõe uma linha ferroviária com 435,854 km. Deseja-se avaliar o atraso total sofrido (soma do atraso de todos os trens) na linha, considerando o despacho de 2, 4, 6, 8, 10 e 12 trens e uma linha de 1, 3, 5 e 7 pátios intermediários (Figura 5.32). Os trens partem em intervalos iguais de tempo, metade deles no sentido *inbound* e a outra metade no *outbound*. Com estas características, criou-se 24 cenários, sendo realizadas 10 simulações para cada um deles. A Figura 5.33 resume os resultados obtidos.

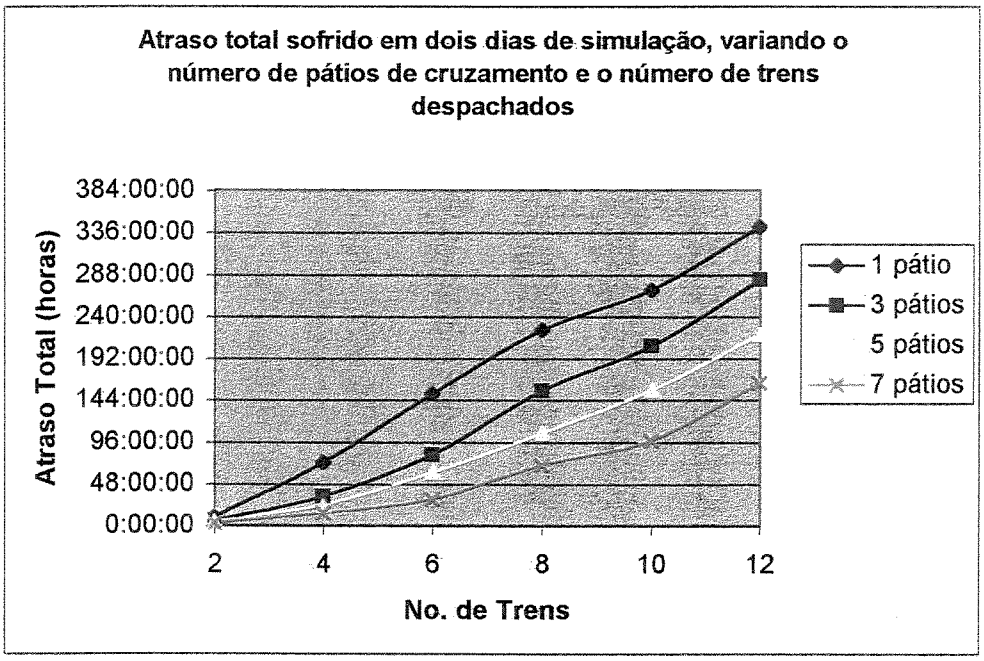


Figura 5.33 Resultados: Exemplo configuração da linha

Analisando os resultados, pode-se observar que conforme se incrementa o número de trens despachados se incrementa o atraso total sofrido. Também observa-se que construindo apenas um pátio de cruzamento intermediário o nível de atraso atingido é muito alto. Este nível de atraso diminui conforme cresce o número de pátios intermediários. Sendo que, o menor atraso é obtido construindo 7 pátios de cruzamento. O usuário do sistema pode usar estes dados para conhecer o benefício a obter construindo pátios intermediários na linha.

5.2.2. Exemplo: Avaliar performance de tipos de trens

Este exemplo ilustra uma análise típica de performance de modelos de trens, considerando a linha ferroviária com 7 pátios.

Os modelos de trens usados no exemplo anterior percorrem os diferentes trechos em um certo tempo. Tomando como base estes modelos, foram definidos modelos de trens dependentes, alguns deles mais rápidos do que os modelos originais e outros mais lentos. Supõe-se que o modelo original contém 8 vagões, e um trem com 6 vagões que tenha tempo de percurso 0.75 (fator de ajuste de tempo) menor que o modelo original, que com 4 seja 0.5 menor, que com 2 seja 0.25 menor e o que com 10 tenha o tempo 1.25 maior. Os trens despachados são seqüenciados em intervalos iguais de tempo, com a metade deles viajando (inicialmente) no sentido *inbound* e a outra metade no sentido *outbound*. A Tabela 5.1 mostra os resultados obtidos neste exemplo.

Fator de Ajuste	**Número de Trens**					
	2	**4**	**6**	**8**	**10**	**12**
0.25 (2 vagões)	2:34:45	7:38:30	11:54:45	25:52:15	42:40:00	60:21:45
0.5 (4 vagões)	3:12:30	12:04:30	23:20:30	43:26:30	91:49:16	119:10:00
0.75 (6 vagões)	3:04:15	11:24:30	21:20:15	60:10:00	95:17:30	148:35:30
1 (8 vagões)	4:00:00	15:45:00	31:59:00	70:40:00	98:26:00	164:54:00
1.25 (10 vagões)	3:41:45	13:41:30	35:54:30	72:48:00	118:03:30	161:43:30

Tabela 5.1 Resultados: Exemplo performance dos trens

Com estes resultados, pode-se concluir que é melhor despachar 2 trens com 4 vagões (atraso de 3:12:30) que despachar 4 trens com 2 vagões (atraso de 7:38:30); também que é melhor despachar 2 trens com 8 vagões (atraso de 4:00:00) que despachar 4 trens com 4 vagões (atraso de 12:04:30).

5.2.3. Exemplo: Avaliar programação de trens

Este exemplo visa mostrar o uso do sistema para avaliar diferentes horários de partida dos trens. O exemplo supõe uma linha ferroviária com 7 pátios de cruzamento (Figura 5.32). Deseja-se despachar 4 trens: dois no sentido *inbound* e dois no *outbound* e quer-se determinar qual é o melhor tempo para despachá-los. Para isto, foram simulados despachos com 4, 8, 12 e 16 horas de intervalo entre cada um deles. Os resultados obtidos apresentam-se na Figura 5.34.

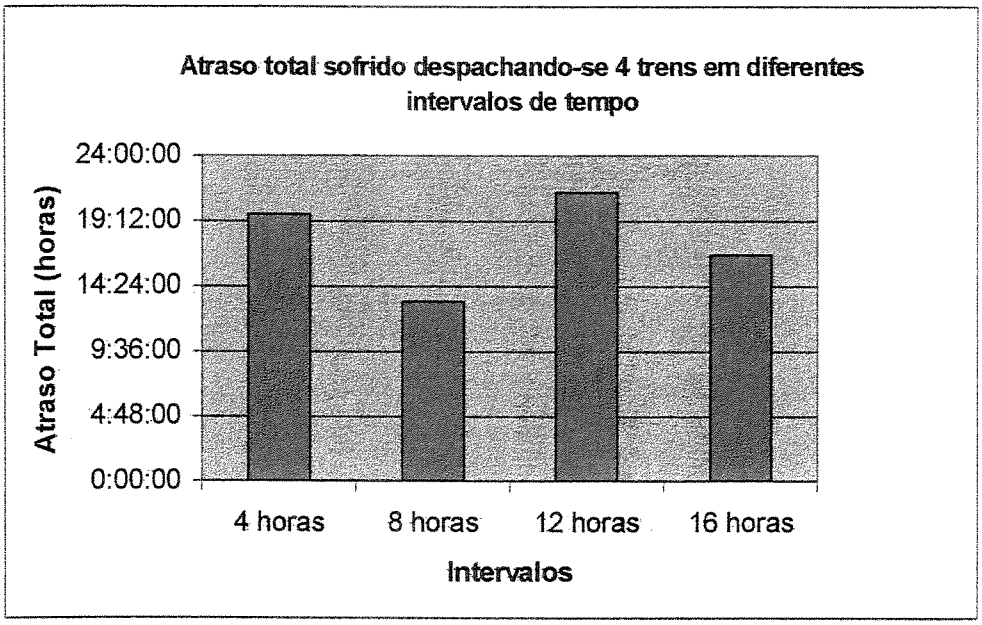


Figura 5.34 Resultados: Exemplo programação de trens

Segundo indica os resultados obtidos, é melhor despachar os trens a cada intervalo de 8.

5.2.4. Exemplo: Avaliar horário de manutenção

Este exemplo objetiva ilustrar a capacidade do sistema no auxilio ao planejamento de atividades de manutenção. Como anteriormente, o exemplo supõe uma linha ferroviária com 7 pátios. Tem-se que despachar 4 trens: dois no sentido *inbound* e outros dois no *outbound*. Deseja-se determinar qual é o melhor intervalo de tempo para despachar os trens e quando é melhor dar manutenção ao SEG5. Foram simulados despachos com intervalos de 4, 8, 12 e 16 horas entre eles, e supondo manutenção no SEG5 às seis horas (6:00), dez horas (10:00), catorze horas (14:00) e dezoito horas (18:00). Os resultados obtidos estão resumidos na Figura 5.35.

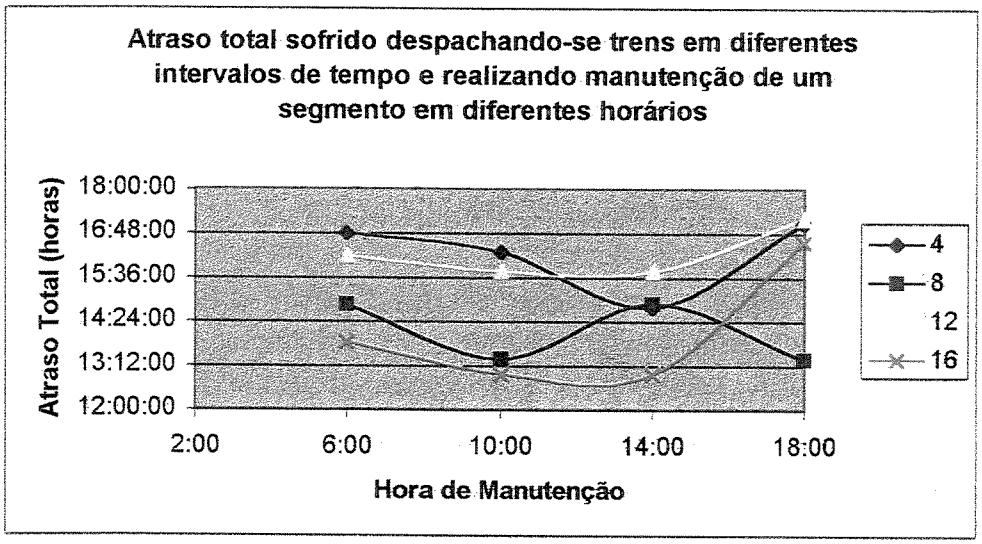


Figura 5.35 Resultados: Exemplo programação de atividades de manutenção

Analisando os resultados, observamos que o menor atraso é obtido despachando os trens com um intervalo de 16 horas e dando manutenção no segmento ou às dez horas (10:00) ou às catorze horas (14:00). Se prefere-se despachar os trens a cada oito horas, a melhor opção para manutenção do segmento SEG5 é ou às dezoito horas (18:00) ou às dez horas (10:00).

5.2.5. Exemplo: Avaliar estratégias de despacho

Este exemplo ilustra a avaliação de três funções de otimização de despacho. A primeira função dá preferência a trens com maior prioridade. A segunda função dá preferência a trens com o maior atraso. A terceira função fornece uma prioridade ao trem de acordo com o atraso. Isto é realizado usando conjuntos e regras nebulosas. Estas funções foram escritas na linguagem definida e são mostradas na seção 3.4.2. Foram realizadas simulações para uma linha com 7 pátios, variando o número de trens e as funções de otimização usadas. Os resultados estão resumidos na Figura 5.36.

Analisando estes resultados, podemos observar que a função que diminuiu o atraso é a que emprega regras e conjuntos nebulosos (fuzzy).

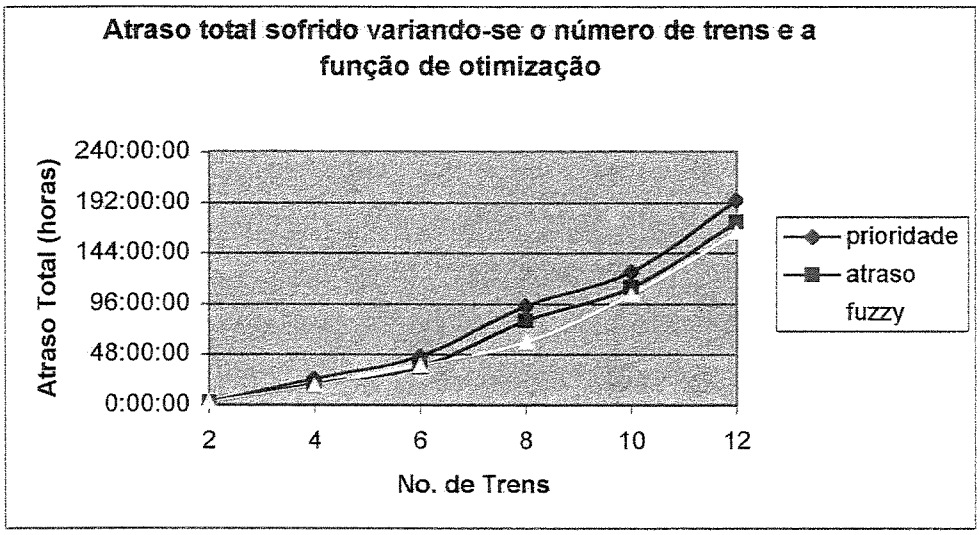


Figura 5.36 Resultados: Exemplo avaliação de estratégias de despacho

5.3. Resumo

Neste capítulo apresentou-se o sistema computacional desenvolvido a partir do modelo de linha proposto e mostrou-se exemplos de seu uso. Este sistema fornece ao usuário a infra-estrutura necessária para simular a circulação de trens em uma malha ferroviária.

O sistema computacional permite definir, de forma amigável, os componentes do modelo de linha proposto: malha ferroviária, modelos de trens, modelos de percurso, modelos de escalas, atividades de manutenção, perturbações dos trens, etc. Dados que já foram inseridos pelo usuário não precisam ser novamente digitados. Quando um componente é definido a partir de um outro componente definido previamente, uma lista de opções é apresentada ao usuário, evitando assim erros desnecessários. Quando os dados necessários não são fornecidos, o sistema apresenta uma mensagem de aviso e encaminha o usuário para preencher os dados que faltam.

O sistema computacional permite a escolha do horizonte de simulação, o número de vezes a simular, a função de otimização usada, etc. Permite também a escolha dos resultados desejados. Esta evita tempo de simulação desnecessário. O sistema facilita diferentes análises: performance dos trens, capacidade de linha, atividades de manutenção, etc.

Foram apresentados cinco exemplos com o intuito de se ilustrar as múltiplas aplicações que podem ser realizadas.

Capítulo 6

Conclusões

Neste trabalho foi proposto um modelo de linha baseado no modelo de Petersen e Taylor [PET82]. Este modelo foi estruturado (considerando o paradigma orientado a objetos) e implementado (usando a linguagem Java).

O modelo de linha proposto representa de forma clara e estruturada os elementos envolvidos na circulação de trens em uma linha ferroviária. O número de componentes que são contemplados por este modelo é maior que o número de componentes contemplados no modelo original de Petersen e Taylor. Esta característica faz com que o modelo represente formas mais realistas que se apresentam no despacho de trens.

Uma característica deste sistema é a forma que este apresenta suas funcionalidades. As funcionalidades do sistema estão separadas em painéis mostrados em forma ordenada (dados de entrada, simulação, resultados e função de otimização). Os dados de entrada são também apresentados em painéis separados. Estes painéis apresentam-se ordenados para orientar o usuário na forma adequada de atualizar os dados. Em alguns casos, se o usuário tenta transgredir esta ordem, uma mensagem de erro é mostrada orientado-o ao painel no qual tem que inserir os dados necessários. A maioria dos painéis de atualização de dados apresentam as mesmas características. Isto é, apresentam uma ou duas tabelas nas partes centrais e botões nas partes inferiores para permitir a inserção ou remoção de elementos. Quando os dados necessários já foram inseridos, o usuário tem somente que escolhê-los, evitando assim erros de inconsistência de dados e facilitando o uso do sistema.

O sistema permite simular diferentes condições, fornecendo a facilidade de configurar os parâmetros de simulação e escolher os resultados desejados. O fato do sistema permitir escolher os resultados desejados, faz com que o usuário obtenha, de forma mais rápida, aqueles em que está diretamente interessado.

Os resultados obtidos pelas simulações são apresentados de forma similar aos dados de entrada; isto é, via painéis ordenados. Os resultados podem ser armazenados em arquivos de texto para permitir sua análise posterior.

O sistema computacional está integrado com uma ferramenta, que fornece ao usuário a possibilidade definir funções de otimização do despacho e usá-las durante a(s) simulação(ões). Estas funções podem ser criadas a partir de algoritmos convencionais e aqueles baseados na teoria de conjuntos nebulosos. Esta característica fornece flexibilidade e versatilidade para refletir situações mais realistas.

O sistema computacional incorpora um algoritmo eficaz e computacionalmente eficiente para a prevenção de bloqueio.

Finalmente, exemplos de uso foram apresentados com a finalidade de ilustrar as análises que podem ser realizadas com o auxílio do sistema computacional.

O sistema computacional desenvolvido serve como base para a realização dos seguintes trabalhos futuros:

- avaliar o uso de diferentes técnicas de otimização de despacho, fazendo uso da linguagem criada ou, se for necessário, expandindo esta linguagem para permitir a definição de outro tipo de funções;
- expandir o sistema para permitir a análise automática de resultados de diferentes cenários de simulação;
- desenvolver sistemas distribuídos de controle de tráfego de trens.

Referências Bibliográficas

[AMB99] Ambrosio, A. "Técnicas de Planejamento e Controle de Sistemas a Eventos Discretos Aplicadas ao Despacho de Trens". Tese de Mestrado, Faculdade de Engenharia Elétrica e Computação, Universidade Estadual de Campinas. Setembro,1999.

[ALL66] Allman, W. "A computer simulation model of railroad freight transportation system". Int. Proc. 4th. Int. Conf. Ops. Res., pp. 339-351 Wiley, New York, 1966

[ASS80a] Assad A. "Models for rail transportation". Transportation Research A-Pol 14:(3), pp. 205-220, 1980.

[ASS80b] Assad A. "Modeling of rail networks - toward a routing-makeup model". Transport Res B-Meth 14:(1-2), pp. 101-114, 1980.

[BRA96] Brannlund, U., et al. "Railway Timetabling using Lagrangian Relaxation". Transportation Science 32:(4), pp. 358-369, 1996.

[CAR94a] Carey, M. "A model and strategy for train pathing with choice of lines, platforms, and routes" Transportation Research B-Methodology 28:(5), pp. 333-353, 1994.

[CAR94b] Carey, M. "Extending a train pathing model from one-way to 2-way track". Transportation Research B-Methodology 28:(5), pp. 395-400, 1994.

[CAR95] Carey, M. "A Model, Algorithms and Strategy for Train Pathing". J. Operational Research Society 46, pp. 988-1005, 1995.

[CHE90] Chen, B., Harker, P. "Two moments estimation of the Delay on Single Track Rail Lines with Scheduled Traffic", Transportation Science 24, pp. 261-275, 1990.

[ENG77] English G. "An analitical model for the analysis of single track railway capacity". Paper for the Int. Symp. on Travel Supply Models, Montreal, 1977.

[ERI98] Eriksson, HE., Penker, M. "UML Toolkit". Willey Computer Publishing, 1 ed., 1998.

[FEL98] Felleisen, M., Johnson, R. "A Little Java, A Few Patterns". MIT Press, 1998.

[FRA66] Frank, O. "Two-Way Traffic on a Single Line of Railway". Operation Research 14, pp. 801-811, 1966.

[GON00] Gonçalves, R. "Sistemas Inteligentes para Planejamento de Escalas de Equipagens em Sistemas de Transporte com Aplicação a Sistemas Ferroviários". Tese de Doutorado, Faculdade de Engenharia Elétrica e Computação, Universidade Estadual de Campinas. Julio, 2000.

[GOM99] Gomide, F., Bessa, E., Vieira, P. e Neto, Luis. "Railway Dispatch Planning and Control". Proc. de NAFIPS'99, pp. 134-138, 1999.

[GRE88] Greenberg, R., et.al. "Predicting Dispatching Delays on a Low Speed, Single Track Railroad". Transportation Science 22, pp. 31-38, 1988.

[HAL96] Hallowell, S., Harker, P. "Predicting On-Time Line Haul Performance in Scheduled Railroad Operations". Transportation Science 30, pp. 364-378, 1996.

[HAR90] Harker, P., Hong, S. "Two moments estimation of the Delay on Double Track Rail Lines with Scheduled Traffic", Journal Transportation Research Forum 31, pp. 38-49, 1990.

[HAR97] Harel, D., Gery, E. "Executable object modeling with statecharts". IEEE Computer, 30: (7), pp. 31-42, 1997.

[HIG96] Higgins A, Kozan E, Ferreira L "Optimal scheduling of trains on a single line track". Transportation Research B-Methodology 30:(2), pp. 147-161, 1996.

[HIG97] Higgins A, Kozan E, Ferreira L "Modeling the number and location of sidings on a single line railway". Computer Operation Research 24:(3), pp. 209-220, 1997.

[HO97] Ho T, Norton J, Goodman C. "Optimal traffic control at railway junctions", IEE Proceedings Electric Power, vol. 144: (2), pp. 140-148, 1997.

[JIA93] Jia, L. e Zhang X. "Distributed railway traffic control system", China Railway Science 14:(3), pp. 85-96, 1993.

[JIA94] Jia, L. e Zhang X. "Distributed intelligent railway traffic control based on fuzzy decisionmaking", Fuzzy Set and Systems 62, pp. 255-265, 1994.

[JOV91] Jovanovic, Harker P. "Tactical Scheduling of Rail Operations: The Scan I System". Transportation Science 25, pp. 46-64, 1991.

[KRA88] Kraft, E. "Analytical models for Rail Line Capacity Analysis". Journal Transportation Research Forum 29, pp. 153-162,1988.

[KRA91] Kraay, D., Harker, P., Chen, B. "Optimal pacing of trains in freight railroads - model formulation and solution". Operation Research 39:(1), pp. 82-99, 1991.

[KRA95] Kraay, D., Harker, P. "Real-time scheduling of freight railroads". Transportation Research B-Methodology 29:(3), pp. 213-229, 1995.

[MIN94] Min, L., Zhang, X. "Intelligent system railway traffic control with fuzzy logic" Fuzzy Sets and Soft Computing 62, pp. 255-265, 1994.

[NOU97] Nõu, A. "Railway Timetabling – Lagrangian Heuristics, Technical report TRITA/MAT-97-OS12, Royal Institute of Technology, Stockholm, Sweden 1997.

[OZE94] Ozekici S, Sengor S. "On a rail transportation model with scheduled services" Transportation science 28:(3), pp. 246-255, 1994.

[PED98] Pedrycz, W., Gomide, F. "An Introduction to Fuzzy Sets: Analysis and Design", MIT Press, 1998.

[PET73] Petersen E., Fullerton, H. "A network flow model of the Canadian railway system". Proc. Int. Conf. on Trans. Res., pp. 611-620. Transportation Research Forum, Chicago, 1973.

[PET74] Petersen E. "Over the Road Transit Time for a Single Track Railway". Transportation Science 8, pp. 65-74, 1974.

[PET75] Petersen E. "Interference Delays on a Partially Double Tracked Railway with Intermediate Signalling", J. Transp. Res. Forum 16, pp. 55-62, 1975.

[PET82] Petersen E., Taylor A. "A Structured Model for Rail Line Simulation and Optimization" Transportation Science 16:(2), pp.192-205, 1982.

[PET83] Petersen E., Taylor A. "Line Block Prevention in Rail Line Dispatch and Simulation Models", Infor. 21:(1), pp. 46-51, 1983.

[PET86] Petersen E., Taylor A., Martland C. "An Introduction to Computer-Assisted Train Dispatch", Journal of Advanced Transportation 20:(1), pp. 63-72, 1986.

[RAM00] Ramírez, L. "Programação de Escalas usando Algoritmos Evolutivos. Aplicação em Empresas de Transporte Ferroviário". Tese de Mestrado, Faculdade de Engenharia Elétrica e Computação, Universidade Estadual de Campinas.

[RIT90] Ritchie S. "Expert Systems in Transportation" Transportation Research 24A:(1), pp. 1, 1990.

[RON00a] Rondón, M., Gomide, F. "Line Block Analysis In Railway Dispatch And Simulation Systems". 9th IFAC Symposium on Control in Transportation Systems, 2000, pp. 405-409.

[RON00b] Rondón, M., Gomide, F. "Modelagem Estruturada e Sistemas Inteligentes: Uma Aplicação a Sistemas de Transporte Ferroviário – I". Reporte Técnico DCA-RT 03/00. Faculdade de Engenharia Elétrica e Computação, Universidade Estadual de Campinas.

[RON00c] Rondón, M., Gomide, F. "Modelagem Estruturada e Sistemas Inteligentes: Uma Aplicação a Sistemas de Transporte Ferroviário – II". Reporte Técnico DCA-RT 04/00. Faculdade de Engenharia Elétrica e Computação, Universidade Estadual de Campinas.

[RON00d] Rondón, M., Gomide, F. "Railway Simulation and Optimization System". World Automation Congress. Third International Symposium on Soft Computing for Industry. Maui, Hawaii June, 2000.

[SAU83] Sauder, R. and Westerman W. "Computer Aided Train Dispatching: Decision Support through Optimization", Interfaces 13, pp. 24-3, 1983.

[SHI66] Shields C. "Models for railroad terminals" IEEE Trans. Sys. Sci. Cybern. SCC2, pp. 123-127, 1966.

[SZP73] Szpizel, B. "Optimal Train Scheduling on a Single Track Railway", In Operational Reaserch `72, M. Ross (ed.) North-Holland, Amsterdam, pp. 343-352, 1973.

[TAH88] Taha, Hamdy A. "Simulation Model and SIMNET", Prentice Hall International Series in Industrial and Systems Engineering, 1988.

[THO71] Thomet M. "A user-oriented freight railroad operating policy". IEEE Trans. Sys. Man. Cyber., SMC-1, pp. 349-359, 1971.

[THO81] Thorndyke P., McArthur D., Cammarata S., e Steeb R., "Distributed Problem solving in air traffic control", in Rep. Second Workshop on Distributed AI, pp. 15-16, June 1981.

[TSU88] Tsurata S., et. al., "A knowledge-based interactive train scheduling system aiming at large scale complex planning expert system" Proc. of IEEE Workshop on AI for Industrial Applications, pp. 490-495,1988.

[UML97] Booch, Grady; Jacobson, Ivar; Rumbaugh, James. Unified Modeling Language Version 1.1 - Setembro, 1997. UML Proposal to the Object Management Group, UML Summary,UML Notation Guide, UML Semantics, UML Extention for Objectory Process for Software Engineering, UML Extention for Busines Modeling, UML Object Constraint Language Specification.

[VER90] Vernazza G., Zunino R. "A distributed intelligence methodology for railway traffic control", IEEE Transactions on Vehicular Technology 39:(3), pp. 263-270, 1990

[VIE96] Vieira, P., Gomide, F. "Computer-aided train dispatch." IEEE Spectrum 33(7), pp.51-53, 1996.

Apêndice

Notação B.N.F. da linguagem desenvolvida

```
CompilationUnit          ::= ( VarDeclaration ";" | Statement )* <EOF>
VarDeclaration           ::= ( "boolean" | "int" | "double" | "Matrix" | "Universe" | "RealTrain" )
                             <IDENTIFIER>
Expression               ::= Assignment
                           | ConditionalOrExpression
Assignment               ::= PrimaryExpression "=" Expression
ConditionalOrExpression  ::= ConditionalAndExpression ( "||" ConditionalAndExpression )*
ConditionalAndExpression ::= InclusiveOrExpression ( "&&" InclusiveOrExpression )*
InclusiveOrExpression    ::= ExclusiveOrExpression ( "|" ExclusiveOrExpression )*
ExclusiveOrExpression    ::= AndExpression ( "^" AndExpression )*
AndExpression            ::= EqualityExpression ( "&" EqualityExpression )*
EqualityExpression       ::= RelationalExpression ( "==" RelationalExpression | "!="
                             RelationalExpression )*
RelationalExpression     ::= AdditiveExpression ( "<" AdditiveExpression | ">" AdditiveExpression |
                             "<=" AdditiveExpression | ">=" AdditiveExpression )*
AdditiveExpression       ::= MultiplicativeExpression ( "+" MultiplicativeExpression | "-"
                             MultiplicativeExpression )*
MultiplicativeExpression ::= UnaryExpression ( "*" UnaryExpression | "/" UnaryExpression | "%"
                             UnaryExpression )*
UnaryExpression          ::= "~" UnaryExpression
                           | "!" UnaryExpression
                           | PrimaryExpression
PrimaryExpression        ::= <IDENTIFIER> "[" ( <INTEGER_LITERAL> | <IDENTIFIER> ) "]"
                           | Literal
                           | Id
                           | "(" Expression ")"
Id                       ::= <IDENTIFIER>
Literal                  ::= TrainListSizeNode
                           | IntConstNode
                           | DoubleConstNode
                           | BooleanLiteral
                           | MatrixLiteral
                           | FuzzySetLiteral
                           | FuzzyPartitionLiteral
                           | UniverseLiteral
                           | MembershipLiteral
                           | FunctionLiteral
                           | TrainPriorityNode
                           | TrainDelayNode
```

```
                    | TimeToCompleteNode
                    | NumberOfReverseTrainsNode
                    | RealTrainNode
IntConstNode        ::= <INTEGER_LITERAL>
DoubleConstNode     ::= <DOUBLE_LITERAL>
BooleanLiteral      ::= "true"
                    | "false"
MatrixLiteral       ::= "[" ( DoubleConstNode | IntConstNode ) ( "," ( DoubleConstNode |
                        IntConstNode ) )* "]"
FuzzyPartitionLiteral ::= "(" FuzzySet ( "," FuzzySet )* ")"
FuzzySet            ::= <IDENTIFIER>
UniverseLiteral     ::= <IDENTIFIER> ".Universe"
FuzzySetLiteral     ::= <IDENTIFIER> "." <IDENTIFIER>
MembershipLiteral   ::= TriangLiteral
                    | TrapeseLiteral
TriangLiteral       ::= ( "triang(" ( DoubleConstNode | IntConstNode ) "," ( DoubleConstNode |
                        IntConstNode ) ")" | "triang(" ( DoubleConstNode | IntConstNode ) "," (
                        DoubleConstNode | IntConstNode ) "," ( DoubleConstNode |
                        IntConstNode ) ")" )
TrapeseLiteral      ::= "trapese(" ( DoubleConstNode | IntConstNode ) "," ( DoubleConstNode |
                        IntConstNode ) "," ( DoubleConstNode | IntConstNode ) "," (
                        DoubleConstNode | IntConstNode ) ")"
Statement           ::= ";"
                    | RuleStatement
                    | IfStatement
                    | LabeledStatement
                    | Block
                    | StatementExpression
                    | WhileStatement
                    | IOStatement
                    | DoWhileStatement
                    | ForStatement
LabeledStatement    ::= <IDENTIFIER> ":" Statement
Block               ::= "{" ( Statement )* "}"
StatementExpression ::= Assignment ";"
IfStatement         ::= "if" "(" Expression ")" Statement ( "else" Statement )?
WhileStatement      ::= "while" "(" Expression ")" Statement
DoWhileStatement    ::= "do" Statement "while" "(" Expression ")"
ForStatement        ::= "for" <IDENTIFIER> "=" ( <INTEGER_LITERAL> | <IDENTIFIER>
                        ) ":" ( <INTEGER_LITERAL> | <IDENTIFIER> ) ( ":"
                        <INTEGER_LITERAL> )? Statement
IOStatement         ::= ReadStatement
                    | WriteStatement
ReadStatement       ::= "read" <IDENTIFIER>
WriteStatement      ::= "write" <IDENTIFIER>
RuleStatement       ::= "if" AntecedentList "then" ConsequentList
AntecedentList      ::= Proposition ( "and" Proposition )*
```

```
Proposition          ::= Id "is" FuzzySetLiteral
ConsequentList       ::= Proposition ( "and" Proposition )*
FunctionLiteral      ::= ASinLiteral
                       | SinLiteral
                       | CosLiteral
                       | ACosLiteral
                       | MinLiteral
                       | MaxLiteral
                       | TanLiteral
                       | ATanLiteral
                       | RunLiteral
                       | expLiteral
                       | logLiteral
                       | sqrtLiteral
                       | powLiteral
ASinLiteral          ::= "asin(" Expression ")"
SinLiteral           ::= "sin(" Expression ")"
CosLiteral           ::= "cos(" Expression ")"
ACosLiteral          ::= "acos(" Expression ")"
TanLiteral           ::= "tan(" Expression ")"
ATanLiteral          ::= "atan(" Expression ")"
MinLiteral           ::= "min(" Expression ( "," Expression )+ ")"
MaxLiteral           ::= "max(" Expression ( "," Expression )+ ")"
expLiteral           ::= "exp(" Expression ")"
logLiteral           ::= "log(" Expression ")"
sqrtLiteral          ::= "sqrt(" Expression ")"
powLiteral           ::= "pow(" Expression "," Expression ")"
RunLiteral           ::= "run(" IntConstNode "," IntConstNode ")"
SelectedTrainNode    ::= "SELECTED_TRAIN"
RealTrainNode        ::= ( "TRAINS[" ( <INTEGER_LITERAL> | <IDENTIFIER> ) "]" |
                       SelectedTrainNode )
TrainListSizeNode    ::= "TRAINS.SIZE"
TrainPriorityNode    ::= ( RealTrainNode | Id ) ".PRIORITY"
TrainDelayNode       ::= ( RealTrainNode | Id ) ".DELAY"
NumberOfReverseTrainsN
ode                  ::= ( RealTrainNode | Id ) ".NUMBER_OF_REVERSE_TRAINS"
NumberOfForwardTrainsN
ode                  ::= ( RealTrainNode | Id ) ".NUMBER_OF_FORWARD_TRAINS"
NumberOfTrainsNode   ::= ( RealTrainNode | Id ) ".NUMBER_OF_TRAINS"
TimeToCompleteNode   ::= ( RealTrainNode | Id ) ".TIMETOCOMPLETE(" SegmentNode ")"
SegmentNode          ::= "SEGMENT"
```