FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Editor Gráfico da Rede Eléctrica - Conectividade

João Paulo Castro Mendes
Leic 2005

U.PORTO

Faculdade de Engenharia da Universidade do Porto

Licenciatura em Engenharia Informática e Computação

Editor Gráfico da Rede Eléctrica – Conectividade na EFACEC Sistemas de Electrónica, S.A.

Relatório do Estágio Curricular da LEIC 2005/06

João Paulo Castro Mendes

Orientador na FEUP: Prof. Jorge Barbosa

Orientador na EFACEC: Pedro Silva

Fevereiro de 2006

For as long as space endures
And for as long as living beings remain,
Until then may I too abide
To dispel the misery of the world.

Dalai Lama, Buddha of Compassion

Resumo

A EFACEC SE (EFACEC Sistemas de Electrónica, S.A.) encontra-se a desenvolver um sistema SCADA/DMS (*Supervisory Control and Data Acquisition / Distribution Management System*), sistema este que permite o controlo e gestão de redes eléctricas. Este sistema possui um modelo completo da rede eléctrica, incluindo a topologia da rede, sobre o qual se podem correr aplicações de cálculo eléctrico.

Até à data, este modelo da rede eléctrica podia ser obtido por importação de dados existentes nos sistemas das empresas de distribuição – o que acarreta uma dependência de outros produtos, ou por introdução manual dos dados em tabelas – que para além de moroso é propenso a erros. De modo a evitar estes inconvenientes, foi decidido desenvolver-se um Editor/Configurador da rede eléctrica, que através de interacção gráfica fosse capaz de capturar a conectividade da rede eléctrica, ou seja, capturar as ligações existentes entre os diversos equipamentos da rede.

Um dos módulos do Editor/Configurador – o Editor Gráfico, tem como objectivo a edição gráfica do modelo da rede eléctrica, usando diagramas onde os diversos equipamentos são representados. De entre as muitas funcionalidades que este módulo deve suportar, destaca-se a capacidade de o utilizador estabelecer e remover ligações entre os diversos equipamentos, ou seja, capturar graficamente a conectividade. O objectivo deste estágio é desenvolver uma solução que cumpra os requisitos relacionados directa ou indirectamente com esta questão da conectividade.

Agradecimentos

Na EFACEC SE quero agradecer ao meu orientador Pedro Silva pelo acompanhamento e apoio que me deu. Quero ainda agradecer aos meus amigos Eduardo Ramalho, Paulo Santos, Paulo Aguiar, Filipe Marinho, José Fernandes e Rui Barreira por todo o apoio que me deram e bons momentos que me proporcionaram.

Na FEUP quero agradecer ao meu orientador, o professor Jorge Barbosa, pela sua ajuda e tempo disponibilizado. Agradeço ainda ao professor Raul Vidal e ao secretariado da LEIC.

Os agradecimentos mais importantes vão para os meus pais, a minha irmã, o meu sobrinho, o Jorge e a minha mais que tudo, a Teresa. Sem eles nada disto seria possível e muito menos seria motivante.

Finalmente quero agradecer o financiamento do PRODEP.

Índice de Conteúdos

1 Introdução

Este capítulo faz uma introdução ao tema do estágio curricular, começando por descrever a empresa onde este decorreu, o problema tratado, os objectivos e o planeamento do trabalho do estágio. Para além disso faz uma pequena descrição do que será abordado nos capítulos que se seguem.

1.1 EFACEC Sistemas de Electrónica, S. A.

A EFACEC é o maior grupo electromecânico Português, com uma facturação anual de 300 milhões de euros e presente em diversos mercados internacionais.

O Grupo EFACEC está espalhado pelo mundo com representação em vários países através de filiais, agências ou através de joint-venture com empresas locais.

A EFACEC Sistemas de Electrónica, S.A., é uma empresa do grupo EFACEC, constituída em 1991 como empresa autónoma, tendo surgido como uma evolução lógica da anterior Electrónica Industrial, criada em 1979 dentro da EFACEC Empresa Fabril de Máquinas Eléctricas, S.A.R.L.

A empresa desenvolve a sua actividade em diversas áreas de negócio que adoptam como base as Tecnologias de Informação e a Electrónica, fazendo uso intensivo de Sistemas de Telecomunicações. A cada área de negócio corresponde, do ponto de vista de organização, uma unidade:

- **Sistemas de Energia, Automação e Telecontrolo**

  Esta unidade dedica-se à automação de estações e subestações (no que diz respeito à distribuição e transporte de electricidade), bem como a Sistemas de Gestão e Supervisão para redes eléctricas e de água.

- **Sistemas de Transporte**

  A unidade de sistemas de transporte trata da sinalização dos transportes ferroviários, bem como das necessidades dos caminhos-de-ferro e estradas no que diz respeito à gestão e sistemas de suporte de informação pública.

- **Sistemas de Fornecimento de Energia**

  A unidade de sistemas de fornecimento de energia desenvolve, comercializa e constrói UPS's (unninterruptible power supplie's) e baterias, assim como rectificadores e conversores de energia eléctrica para aplicações especiais.

- **Soluções Integradas**

  A unidade de soluções integradas fornece e instala infra-estruturas para telecomunicações integradas para os operadores e para as redes dedicadas nos sistemas dos caminhos-de-ferro e estradas.

Este projecto (Editor Gráfico da Rede Eléctrica - Conectividade), insere-se num estágio curricular proposto pela EFACEC Sistemas de Electrónica, S.A., a decorrer no Departamento de Gestão de Redes, Unidade de Automação de Sistemas de Energia.

1.2 Problema

A EFACEC SE (EFACEC Sistemas de Electrónica, S.A.) encontra-se a desenvolver um sistema SCADA/DMS (*Supervisory Control and Data Acquisition / Distribution Management System*), sistema este que permite o controlo e gestão de redes eléctricas. Este sistema possui um modelo completo da rede eléctrica, incluindo a topologia da rede, sobre o qual se podem correr aplicações de cálculo eléctrico.

Até à data, este modelo da rede eléctrica podia ser obtido por importação de dados existentes nos sistemas das empresas de distribuição – o que acarreta uma dependência de outros produtos, ou por introdução manual dos dados em tabelas – que para além de moroso é propenso a erros. De modo a evitar estes inconvenientes, foi decidido desenvolver-se um Editor/Configurador da rede eléctrica, que através de interacção gráfica fosse capaz de capturar a conectividade da rede eléctrica, ou seja, capturar as ligações existentes entre os diversos equipamentos da rede.

1.3 Objectivos

Um dos módulos do Editor/Configurador – o Editor Gráfico, tem como objectivo a edição gráfica do modelo da rede eléctrica, usando diagramas onde os diversos equipamentos são representados. De entre as muitas funcionalidades que este módulo deve suportar, destaca-se a capacidade de o utilizador estabelecer e remover ligações entre os diversos equipamentos, ou seja, capturar graficamente a conectividade. O objectivo deste estágio é desenvolver uma solução que cumpra os requisitos relacionados directa ou indirectamente com esta questão da conectividade.

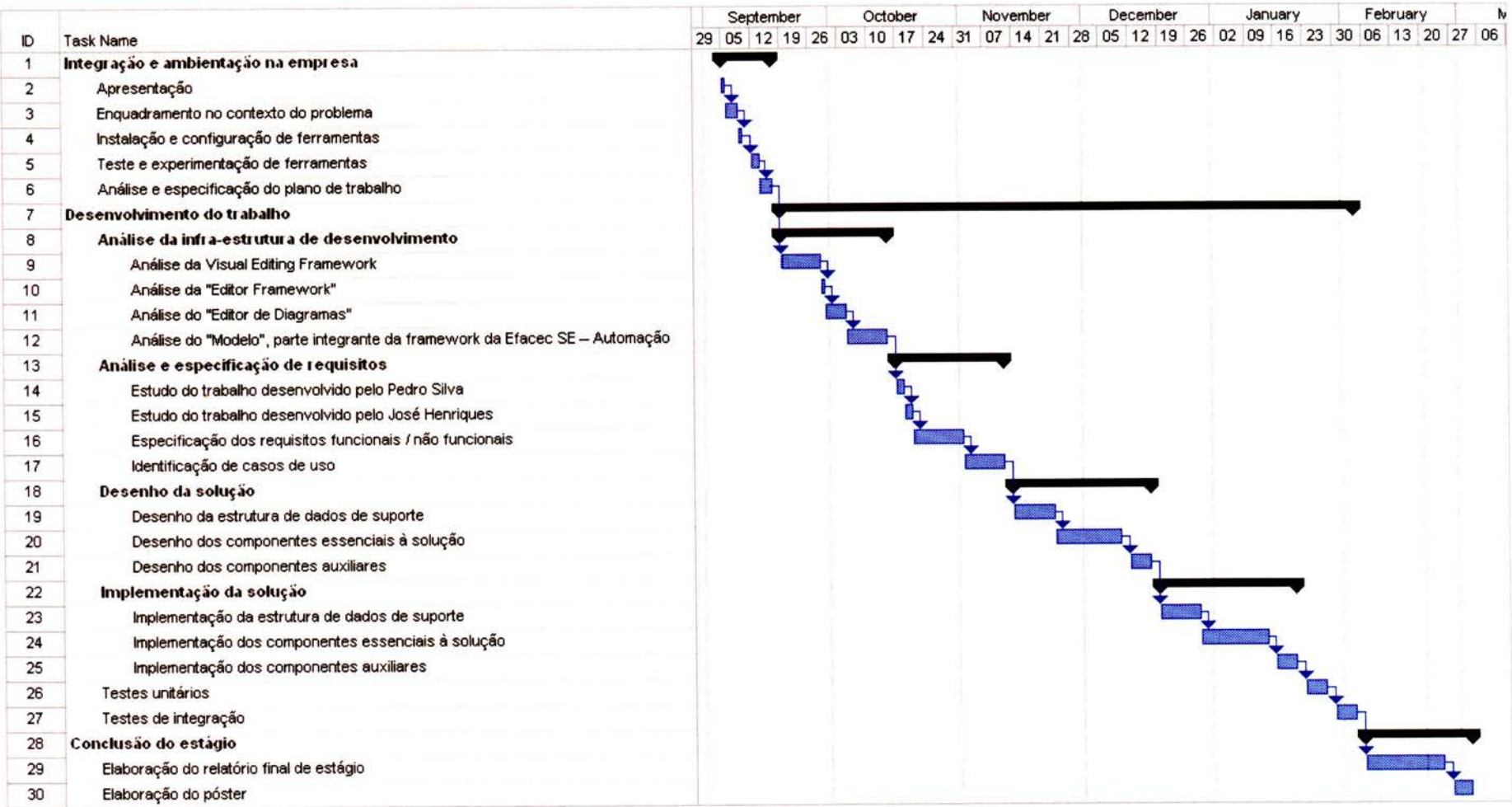
1.4 Plano

O estágio curricular foi projectado para uma duração de cinco meses, tendo quatro etapas essenciais: Análise, Implementação, Validação e Documentação.

Ao longo do projecto, houve a necessidade de efectuar algumas alterações no planeamento, devido:

- Ao menor nível de conhecimento do problema em questão, na altura do planeamento;
- A falhas na estimativa do tempo das tarefas; e
- Ao aparecimento de novas tarefas não tomadas em conta no primeiro planeamento.

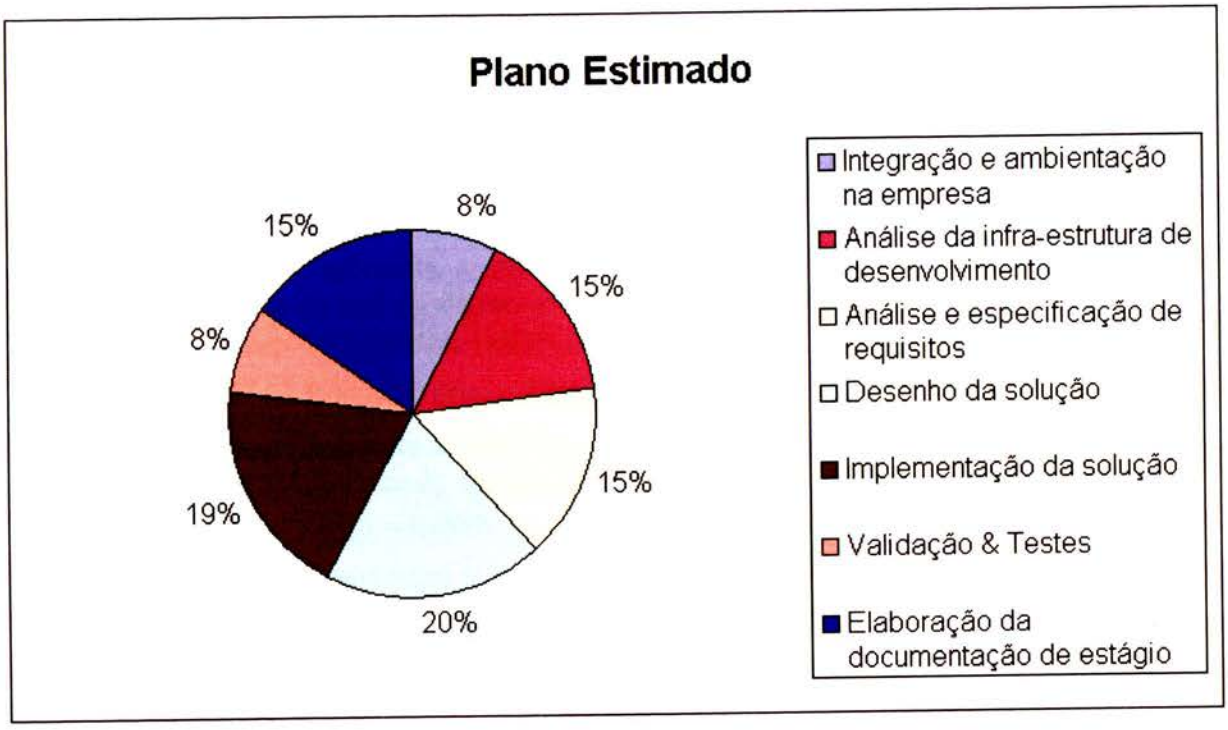
O projecto foi desenvolvido na linguagem *Java* utilizando o ambiente *Eclipse*. O repositório de dados utilizado foi o *MS Visual SourceSafe*.

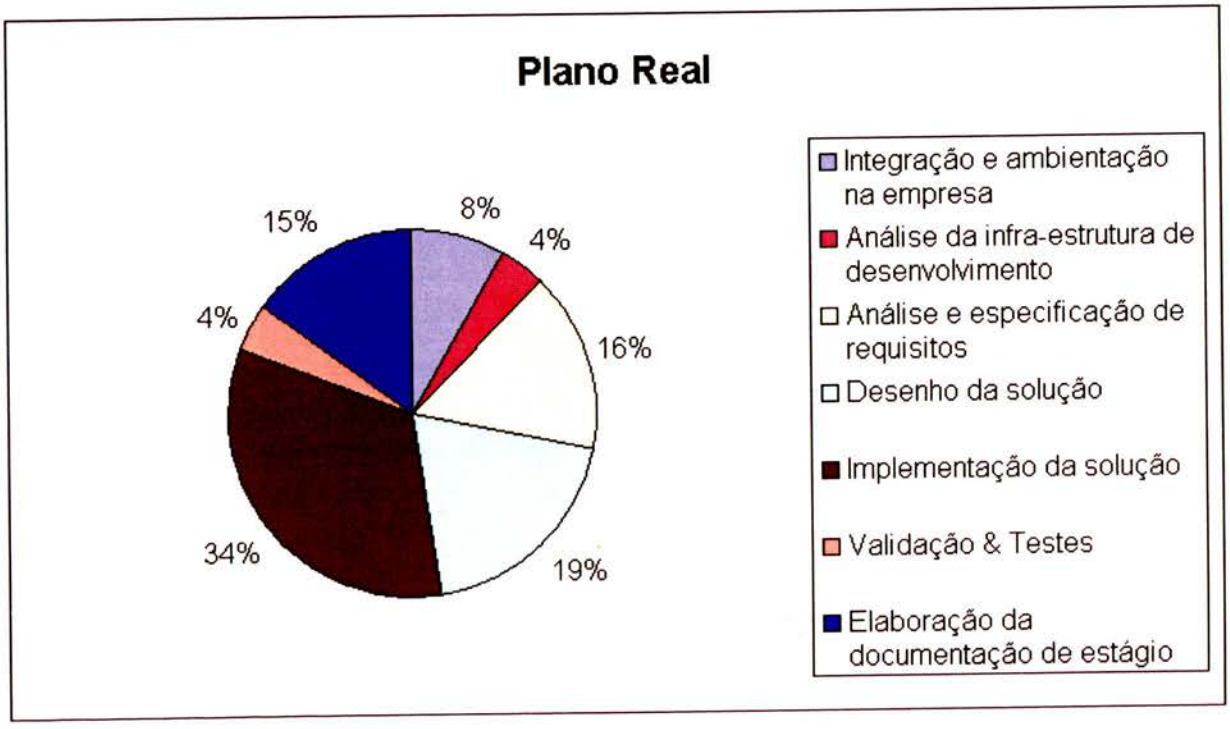


ID	Task Name
1	**Integração e ambientação na empresa**
2	Apresentação
3	Enquadramento no contexto do problema
4	Instalação e configuração de ferramentas
5	Teste e experimentação de ferramentas
6	Análise e especificação do plano de trabalho
7	**Desenvolvimento do trabalho**
8	**Análise da infra-estrutura de desenvolvimento**
9	Análise da Visual Editing Framework
10	Análise da "Editor Framework"
11	Análise do "Editor de Diagramas"
12	Análise do "Modelo", parte integrante da framework da Efacec SE – Automação
13	**Análise e especificação de requisitos**
14	Estudo do trabalho desenvolvido pelo Pedro Silva
15	Estudo do trabalho desenvolvido pelo José Henriques
16	Especificação dos requisitos funcionais / não funcionais
17	Identificação de casos de uso
18	**Desenho da solução**
19	Desenho da estrutura de dados de suporte
20	Desenho dos componentes essenciais à solução
21	Desenho dos componentes auxiliares
22	**Implementação da solução**
23	Implementação da estrutura de dados de suporte
24	Implementação dos componentes essenciais à solução
25	Implementação dos componentes auxiliares
26	Testes unitários
27	Testes de integração
28	**Conclusão do estágio**
29	Elaboração do relatório final de estágio
30	Elaboração do póster

Plano Estimado vs. Plano Real

Apresenta-se de seguida a comparação entre o plano estimado e o plano real. A grande diferença reside nas fases "Análise da infra-estrutura de desenvolvimento" e "Implementação da solução".





1.5 Estrutura do Relatório de Estágio

Ao longo dos cinco capítulos deste relatório, será feita a descrição de todo o trabalho realizado para alcançar o objectivo do estágio:

- **Capítulo 1 – Introdução**

  Este capítulo faz uma introdução ao tema do estágio curricular, começando por descrever a empresa onde este decorreu, o problema, os objectivos e o planeamento.

- **Capítulo 2 – O sistema SCADA/DMS**

  Este capítulo descreve alguns conceitos envolvidos neste projecto. É apresentado um diagrama de contexto de forma a facilitar a compreensão global do problema.

- **Capítulo 3 – Visual Editing Framework**

  Este capítulo apresenta a arquitectura da VEF tendo em conta que uma parte significativa do que foi desenvolvido esteve directamente ligada a esta (extensões e especializações de funcionalidades já existentes).

- **Capítulo 4 – Editor Gráfico – Conectividade**

  Este capítulo apresenta os requisitos, arquitectura e alguns detalhes de implementação da solução desenvolvida. Por isso mesmo, representa o trabalho realizado, sendo os anteriores capítulos, capítulos de introdução e contextualização do problema.

- **Capítulo 5 – Conclusões**

  Apresentação das conclusões finais e futuros desenvolvimentos.

2 O sistema SCADA/DMS

Depois de uma pequena introdução ao tema do projecto e aos seus objectivos, iremos nos capítulos seguintes abordar com mais detalhe o problema e a implementação da solução. Mas antes, este capítulo introduz os conceitos que vão permitir uma melhor compreensão dos capítulos que se seguem. Vão ser aqui descritas as tecnologias associadas a este projecto assim como a sua interligação.

2.1 SCADA

Os sistemas SCADA (*Supervisory Control and Data Aquisition*) são responsáveis pela recolha e análise de informação em tempo real. Actualmente estes sistemas recorrem às mais avançadas tecnologias de computação e comunicação para monitorizar e controlar estruturas ou equipamentos industriais dispersos geograficamente e recorrem a interfaces gráficas para tornar a interacção com o utilizador mais amigável.

Este tipo de sistemas é usado em várias indústrias: Telecomunicações, Águas e Saneamento, Energia, Gás, Combustíveis e Transportes.

Hoje em dia estes sistemas são distribuídos, com utilizadores e dispositivos dispersos, dispositivos estes que poderão falhar sem prejudicar o sistema, uma ou mais bases de dados, com suporte para um desenvolvimento contínuo em várias plataformas de computação.

A integração com outros sistemas, novos ou já existentes, é de grande importância. O utilizador deverá ter a percepção de que o sistema trata de um grande volume de informação, mas que não deixa de ser fácil de operar. Os sistemas SCADA devem ser totalmente configurados de forma a poderem responder a todas a especificações do comprador.

2.2 DMS

Os sistemas DMS (*Distribution Management System*) surgiram como uma evolução dos sistemas SCADA usados na supervisão de redes eléctricas.

Os sistemas SCADA monitorizam pontos (equipamentos) da rede que controlam, possibilitando também o seu controlo. Estes sistemas não têm noção da interligação destes pontos.

Os sistemas DMS são sistemas vocacionados para as redes eléctricas, que suportam todas as funcionalidades dos sistemas SCADA e que para além disso dispõem de informação do modelo de conectividade da rede. Esta informação permite disponibilizar uma série de ferramentas que ajudam o controlo da rede eléctrica:

- **Previsor de Cargas**

  Esta ferramenta permite prever o consumo de energia de determinados equipamentos.

- **Fluxo de Cargas**

  Ferramenta que permite calcular, através de modelos matemáticos, o fluxo de energia que passa em determinados pontos da rede.

- **Análise de Curto-circuito**

Ferramenta que estuda o comportamento da rede em determinados cenários.

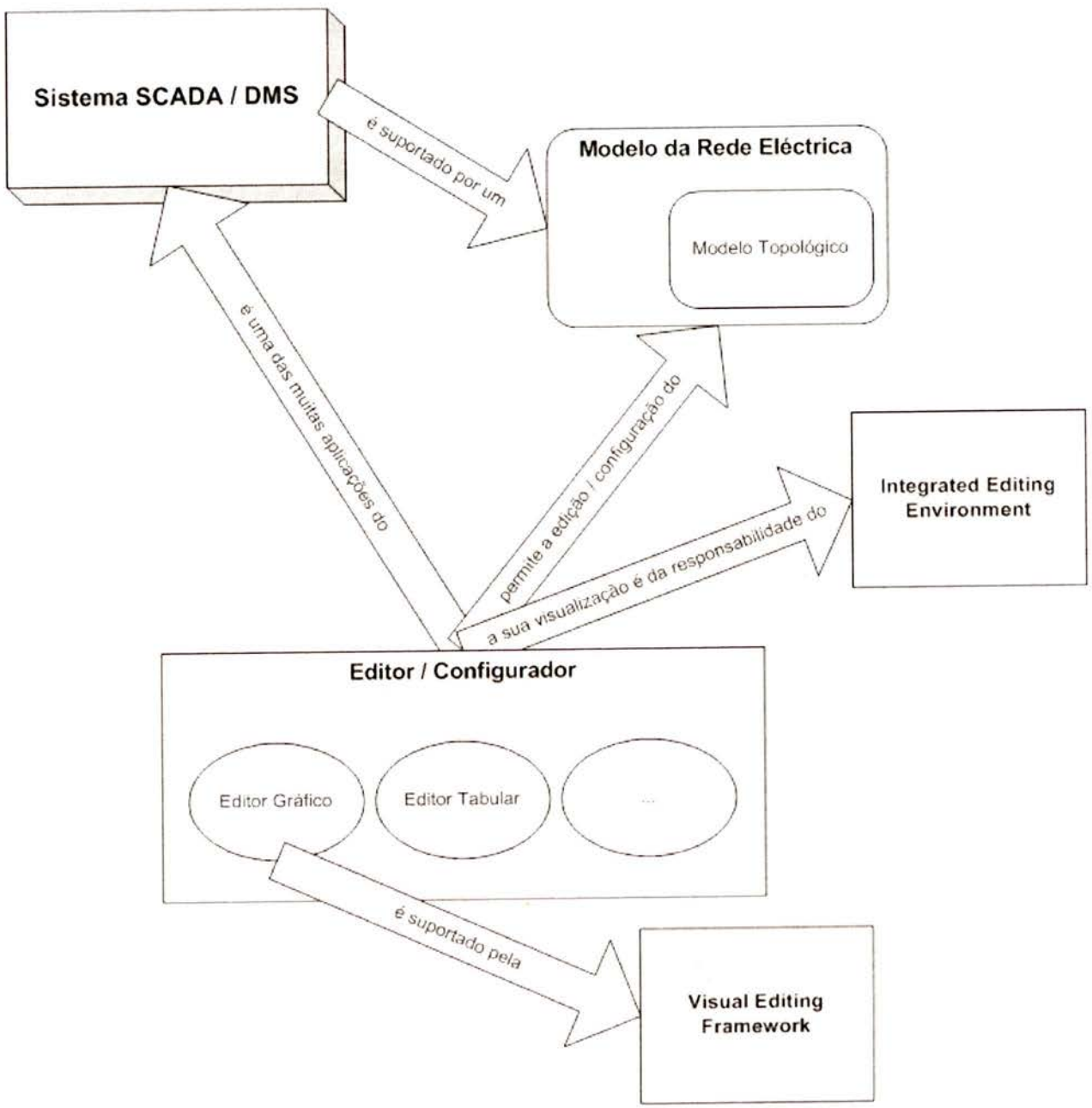
- **Alocação de Cargas**

  Esta ferramenta permite calcular a carga de determinados equipamentos (não monitorizados), com base em medidas monitorizadas pelo sistema.

- **Regras de Segurança**

  Este módulo permite definir algumas regras de segurança nas operações de manutenção ou reparação da rede. Esta ferramenta permite também assegurar e verificar se as regras estão a ser cumpridas.

2.3 Diagrama de Contexto



2.4 Editor/Configurador

O Editor/Configurador, sendo uma das muitas aplicações do SCADA/DMS, tem como objectivo a edição e configuração da rede eléctrica.

É suportado por um Modelo (fonte de dados) e é constituído por diversos módulos, tais como: Editor Gráfico, Editor Tabular, etc.

A sua visualização e respectivo ambiente gráfico são da responsabilidade de uma outra aplicação: o IEE (*Integrated Editing Environment*).

2.5 Editor Gráfico

Como já foi referido, o Editor Gráfico permite a edição gráfica do modelo da rede eléctrica. Sendo um módulo integrante do Editor/Configurador é também suportado pelo Modelo (fonte de dados). Este módulo utiliza ainda a *Visual Editing Framework* como biblioteca de suporte gráfico.

A solução foi desenvolvida neste módulo.

2.6 Visual Editing Framework

A *Visual Editing Framework* é uma biblioteca de suporte gráfico. Entre outras, suporta as seguintes principais funcionalidades:

- Representação de objectos gráficos em vistas/diagramas.
- Selecção simples/múltipla de objectos.
- Transformações sobre objectos (translação, escalamento, rotação, etc.).
- *Zoom & Pan.*

2.7 Modelo

O modelo representa a fonte de dados. Permite carregar e guardar a informação relativa à rede, bem como, *undo* e *redo* de operações. O modelo disponibiliza um protocolo que todos os objectos que necessitem de ser persistentes devem cumprir. Tal facto garante transparência para quem o utiliza.

3 Visual Editing Framework

Tendo em conta o objectivo – captura gráfica de conectividade, e o facto de que a VEF é o suporte gráfico do Editor Gráfico, apresenta-se de seguida uma análise da arquitectura desta biblioteca gráfica.

3.1 Funcionalidade Geral

De uma forma geral, a funcionalidade coberta por esta biblioteca é:

- Objectos gráficos, com geometria e atributos gráficos, que se saibam desenhar em janelas de visualização (incluindo polilinha, elipse, texto, etc.)
- Edição das propriedades gráficas (cor, estilo linha, largura linha, fonte) dos objectos. Estas classes usam uma janela gráfica onde o utilizador pode escolher/definir essas propriedades.
- Símbolos, constituídos por objectos gráficos. Existe também um objecto gráfico, a "colocação de símbolo", que é representada à custa do desenho de um símbolo na posição e tamanho definidos pela colocação.
- Diagramas compostos por objectos gráficos, e janelas onde mostrar esses diagramas (múltiplas janelas para o mesmo diagrama). Navegação nas janelas de diagramas através de acções de rato (*Zoom & Pan*).
- Edição da geometria dos objectos gráficos através de acções de rato, que permitam seleccionar e depois arrastar, rodar, redimensionar, ou de outra forma alterar a geometria dos objectos.

3.2 Modelo de Classes

As classes descritas nesta secção encontram-se implementadas na *package* **pt.efacec.se.aut.tools.vef**. Esta *package* encontra-se organizada da seguinte forma:

- **vef**

  A *package* base. Nesta *package* encontram-se as interfaces e classes base de suporte à VEF. Especializações das classes base, ou classes não fundamentais na arquitectura existem em *sub-packages*.

- **vef.graphic**

  Contém especializações de classes de objectos gráficos e suas selecções (uma selecção é um objecto gráfico usado para representar outro objecto gráfico no estado de selecção).

- **vef.interactor**

  Contém especializações de classes de interactores. Um interactor é uma classe responsável por processar os eventos gráficos que ocorram numa dada vista de um diagrama de modo a alterar os objectos gráficos mostrados nessa vista, ou alterar a própria vista.

- **vef.beans.infos**

Todas as classes de gráficos (e portanto existentes em **vef.graphic**) são *beans* cujas propriedades são *bounded*. Esta *package* contém *BeanInfos* para as classes de gráficos para as quais o *Introspector* não é capaz de gerar um *BeanInfo* adequado.

- **vef.beans.editors**

  Contém implementações de *java.beans.PropertyEditor* para os tipos de propriedades para as quais o *Introspector* não é capaz de encontrar um editor adequado.

- **vef.event**

  Contém a definição das interfaces de *listeners* e de classes de eventos usadas na VEF para notificar alterações nos diversos tipos de modelos.

- **vef.util**

  Contém classes que não são fundamentais na arquitectura da VEF e que não encaixam nas restantes *packages*.

vef

No vef, os objectos gráficos (*Graphic*) existem em camadas (*Layer*) pertencentes a diagramas (*Diagram*). Cada diagrama pode ter várias vistas (*DiagramView*) mostrando os objectos do diagrama, cada vista tendo uma transformação de coordenadas própria sobre o espaço de coordenadas do diagrama, de modo a poder mostrar áreas distintas do diagrama.

Quando um objecto gráfico é seleccionado, é criado um objecto do tipo *Selection* que é composto por um conjunto de *Handles* colocados em posições especiais do objecto seleccionado. A selecção num diagrama é gerida pelo *SelectionManager*, e em geral o utilizador modificará a geometria de um objecto através do arrasto dos *Handles* de selecção desse objecto.

A interacção do utilizador com a vista ou com os objectos gráficos nela visualizados é mediada por instâncias de *Interactor*, que são geridos por um *InteractorManager* – cada vista possui o seu próprio *InteractorManager* sendo portanto possível interagir de maneira diferente em cada uma das vistas.

No vef, todas as alterações relevantes podem ser observadas através de *EventListeners* especializados para cada parte do modelo. Assim, um *DiagramListener* recebe notificações de alterações ao diagrama, aos seus objectos gráficos, ou às suas camadas; um *SelectionListener* recebe notificações sempre que o conjunto de objectos seleccionados de um diagrama é alterado; e um *InteractorListener* recebe uma notificação sempre que um *Interactor* é adicionado ou removido a uma vista.

A classe *AbstractGraphic* é uma implementação abstracta da interface *Graphic*, e oferece suporte para o registo do gráfico num *GraphicsOwner*.

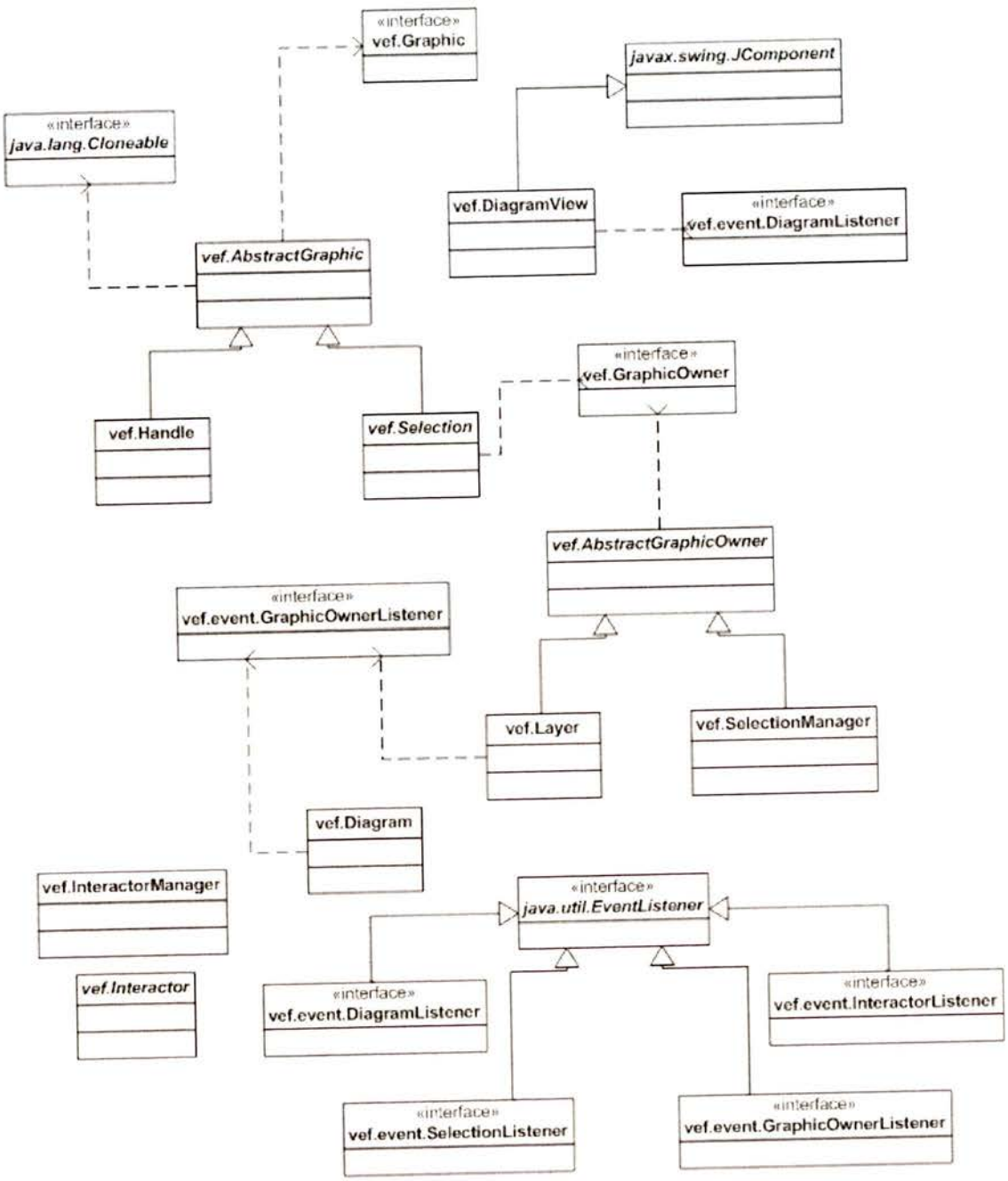
Um *GraphicOwner* é um contentor de gráficos. Qualquer alteração no conteúdo deste contentor (adicionar/remover gráficos) ou qualquer alteração que ocorra num gráfico gerido por este é notificada aos *GraphicOwnerListeners* registados neste *GraphicOwner*.

A classe *AbstractGraphicOwner* é uma implementação abstracta da interface *GraphicOwner* possuindo o mecanismo de suporte de notificações aos *GraphicOwnerListeners*.

Existem várias implementações concretas de *GraphicsOwner*: uma *Layer* é o contentor de objectos de um diagrama, um *SelectionManager* contém gráficos do tipo *Selection* que

representam os gráficos correntemente seleccionados num diagrama, e uma *Selection* é um *GraphicsOwner* de gráficos do tipo *Handle*.

Diagrama de Classes



vef.graphic

A classe *ShapeGraphic* é a classe base de gráficos que especificam a sua geometria a partir de uma *java.awt.Shape*. Um *ShapeGraphic* pode ser desenhado através da linha e da região interior da *shape*, sendo possível definir o valor de várias propriedades relacionadas com estas capacidades: cor de linha, cor de preenchimento, largura de linha, estilo de linha, etc.

A classe *Polyline* pode ser usada para modelar tanto uma polilinha aberta como um polígono fechado. A sua selecção é dada pela classe *PolyPointSelection*.

A classe *RectangularShapeGraphic* aceita como geometria qualquer classe *java.awt.geom.RectangularShape*, o que significa que pode modelar por exemplo rectângulos ou elipses.

A classe *ArcGraphic* é suportada por uma geometria do tipo *java.awt.geom.Arc2D*, que é uma das subclasses de *RectangularShape*. A classe *ArcGraphic* existe sobretudo para permitir ter a sua própria especialização de *Selection* que para além dos *handles* nos cantos da caixa envolvente mostra também um *handle* no ponto de início do arco e no ponto de fim do arco.

A classe *TextGraphic* é desenhada com um texto (multilinha) sobre uma geometria do tipo *RectangularShape* (rectângulo, elipse, arco, etc.). É possível definir qual a fonte a usar durante o desenho do texto, e qual o alinhamento do texto relativamente à geometria rectangular de fundo.

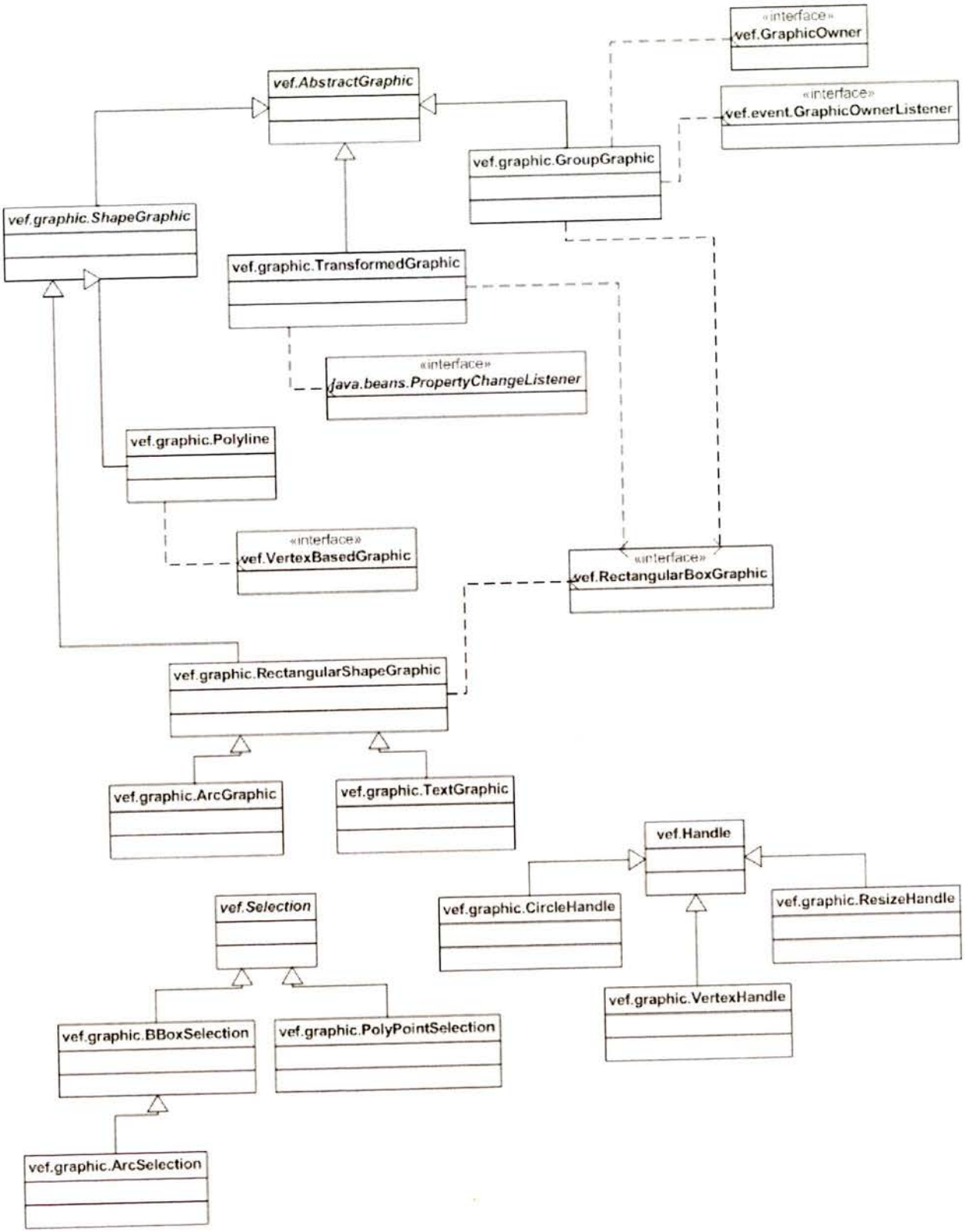
Existem dois casos de gráficos que não são subclasses de *ShapeGraphic* porque a sua geometria não é dada por uma *shape*. São eles *GroupGraphic* e *TransformedGraphic* – o objecto de selecção usado em qualquer um dos casos é um *BBoxSelection*, que mostra *handles* nos cantos da caixa envolvente do grupo ou do gráfico transformado.

Um *TransformedGraphic* é um objecto gráfico que referencia outro objecto gráfico.

Um *GroupGraphic* é um gráfico que é composto por uma colecção de gráficos (podendo incluir outros *GroupGraphics*).

As classes *CircleHandle*, *VertexHandle* e *ResizeHandle* representam respectivamente: handle de início e fim de arco, handle de vértice e handle de redimensionamento.

Diagrama de Classes



vef.interactor

Todas as classes responsáveis pela interacção entre o utilizador e uma vista estendem a classe *Interactor*. Esta classe base declara os métodos que deverão ser redefinidos nas subclasses concretas de interacção de modo a implementar comportamento específico.

Na *package vef.interactor* existem definidas várias subclasses de *Interactor*, que podemos classificar em interactores de vista, interactores de criação de gráficos e interactores de modificação de gráficos.

Interactores de Vista

Um *NavigationInteractor* é responsável por alterar a transformação de coordenadas de uma vista. Este interactor permite fazer *Zoom In*, *Zoom Out* e *Pan*.

Um *SelectionInteractor* é responsável pela selecção. É capaz de selecção simples, remover selecção, múltipla selecção recorrendo a uma tecla, múltipla selecção por definição de uma área e remoção de objectos seleccionados. Se um utilizador clicar num *handle* de um objecto que se encontra seleccionado, um interactor especializado que permite a modificação do objecto é utilizado.

Interactores de Criação de Gráficos

Um *GraphicCreationInteractor* é responsável por criar e adicionar um novo gráfico ao diagrama no ponto onde o utilizador clicar. Este interactor é capaz de colocar qualquer objecto gráfico. No entanto, existem gráficos que necessitam de uma interacção mais elaborada.

Por exemplo, na criação de um gráfico do tipo *TextGraphic* pretende-se que logo a seguir à colocação o gráfico fique imediatamente num estado que permita a introdução de texto – a classe *TextCreationInteractor* resolve este problema estendendo *GraphicCreationInteractor*.

A criação de polilinhas é outro caso que não é convenientemente resolvido através de um *GraphicCreationInteractor*. A classe *PolylineCreationInteractor* resolve apropriadamente este problema começando por criar uma instância de *Polyline* com um único vértice localizado no ponto do click, e em seguida criando uma instância de *PolyPointCreationInteractor* que é responsável por criar os restantes vértices da polilinha.

As classes *LineCreationInteractor* (permite criar linhas) e *PolygonCreationInteractor* (polilinha fechada) são especializações da classe *PolylineCreationInteractor*.

Interactores de Modificação de Gráficos

Os interactores descritos nesta secção são interactores especializados em modificar a geometria (ou outras propriedades) de gráficos já existentes.

Um *MoveInteractor* é um interactor que é capaz de mover (aplicar uma transformação de translação) todos os objectos seleccionados de acordo com o arrasto do rato.

Um *TextInteractor* permite a introdução de texto num *TextGraphic*.

A classe *DragHandleInteractor* é a classe base de interactores cuja funcionalidade é o arrasto de um *handle*.

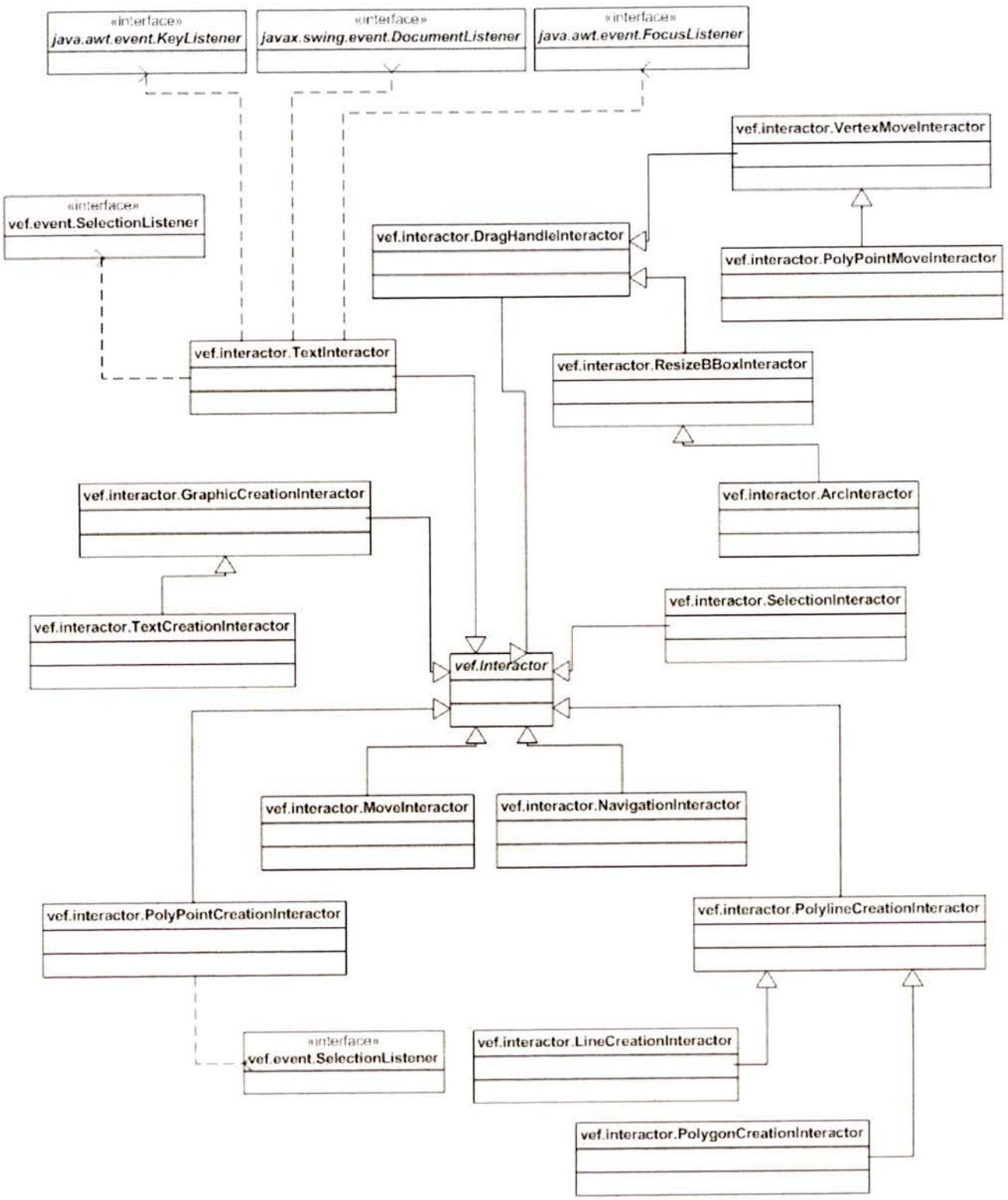
A classe *BBoxResizeInteractor* é um interactor de modificação adequado a objectos gráficos cuja selecção é do tipo *BBoxSelection*, que mostra *handles* nos cantos e a meio dos limites da caixa envolvente. Este interactor permite o redimensionamento dos objectos.

A classe *ArcInteractor* é um interactor adequado a objectos gráficos do tipo *ArcGraphic*. Estende *BBoxResizeInteractor* permitindo que se arraste os *handles* especiais de *ArcSelection* que definem os ângulos de fim e de início do arco.

A classe *VertexMoveInteractor* é um interactor capaz de mover os vértices de objectos gráficos que implementam a interface *VertexBasedGraphic*.

A classe *PolyPointMoveInteractor* estende *VertexMoveInteractor* de modo a deixar criar novos vértices na polilinha.

Diagrama de Classes



4 Editor Gráfico – Conectividade

Após a contextualização do problema e da exposição resumida da arquitectura da VEF, este capítulo apresenta os requisitos, arquitectura e alguns detalhes de implementação da solução desenvolvida.

4.1 Funcionalidade Geral

A funcionalidade será apresentada de uma forma detalhada na secção dos requisitos. No entanto, de uma forma geral, o utilizador deve ser capaz de:

- Capturar graficamente a conectividade de um diagrama num dado instante.
- Interagir com os diferentes tipos de objectos do diagrama de forma a:
  - Criar novas ligações,
  - Remover ligações existentes,
  - Manter as ligações existentes.

4.2 Requisitos

Apresentam-se de seguida os requisitos detalhados da solução desenvolvida.

Tipos de objectos

Num diagrama podem existir os seguintes tipos de objectos:

- **Colocação**: colocação de um símbolo, símbolo este que é uma representação gráfica de um equipamento físico existente na realidade.

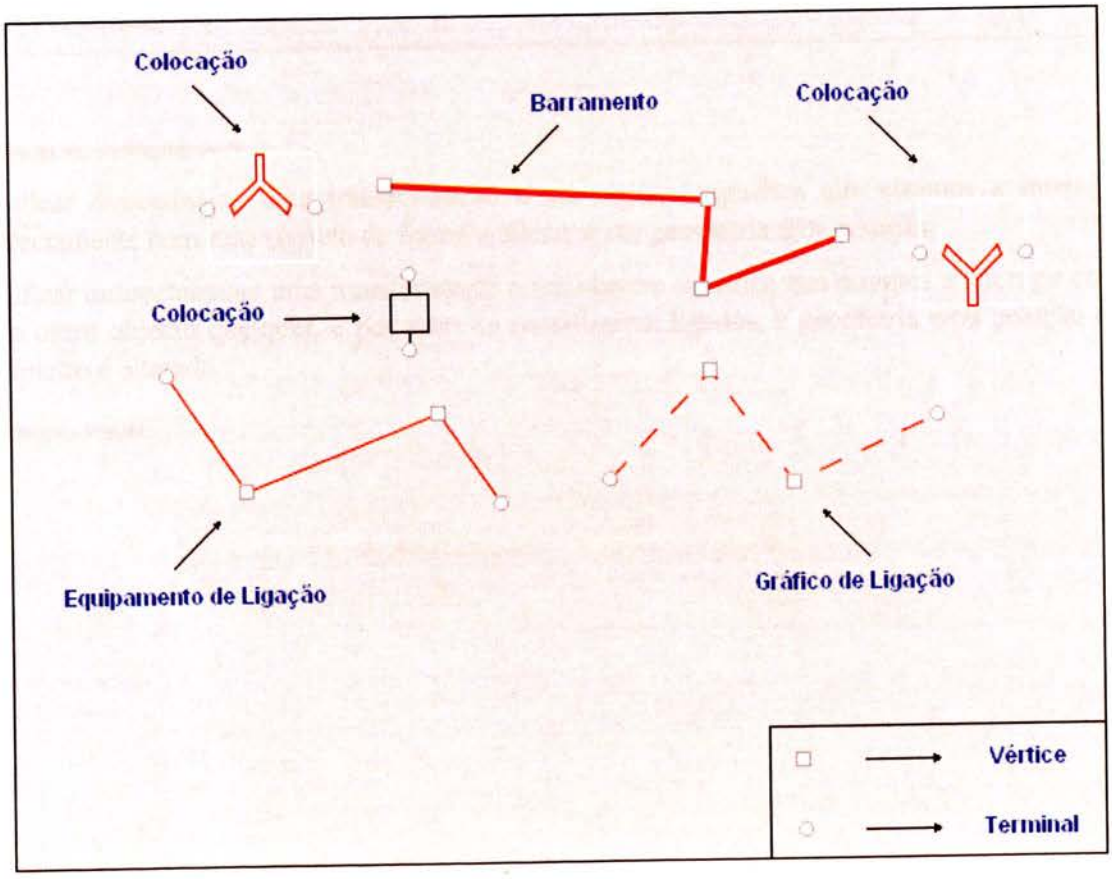
  Exemplo(s): Disjuntor
- **Barramento**: representação gráfica de um equipamento físico existente na realidade.
- **Equipamento de Ligação**: representação gráfica de um equipamento físico existente na realidade.

  Exemplo(s): Cabo de ligação / Troço de linha
- **Gráfico de Ligação**: representação gráfica de uma ligação entre equipamentos físicos existentes na realidade.

Na tabela a seguir enunciam-se as suas características:

	Colocação	Barramento	Equipamento de Ligação	Gráfico de Ligação
Geometria	É dada pela posição dos seus terminais	É dada por um conjunto de vértices (polilinha)	É dada por um conjunto de vértices (polilinha)	É dada por um conjunto de vértices (polilinha)
Nº de terminais	[1; N]	1 (ao longo de toda a sua geometria)	2 (localizados nos extremos)	2 (localizados nos extremos)
Disponível numa palete?	✓	✓	✓	✓

Exemplo visual



Transformações

As transformações disponíveis ao utilizador são:

- **Translação**

  Exemplo(s): mover um barramento

- **Translação de Vértice**

  Exemplo(s): mover um vértice de um barramento

- **Translação de Terminal**

  Exemplo(s): mover um terminal de um disjuntor

Na tabela a seguir mostra-se que transformações podem ser aplicadas aos diferentes tipos de objectos:

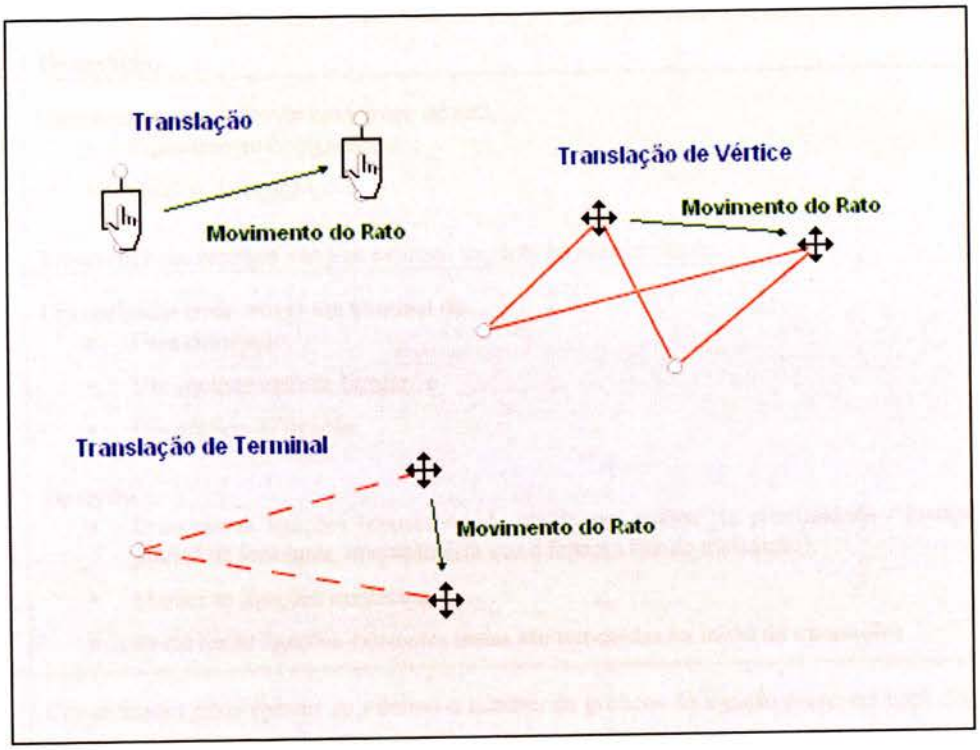
	Colocação	**Barramento**	**Equipamento de Ligação**	**Gráfico de Ligação**
Translação	✓	✓	✓	✓
Translação de Vértice	✗	✓	✓	✓
Translação de Terminal	✓	✗	✓	✓

Directa vs. Indirectamente

Aplicar directamente uma transformação a um objecto significa que estamos a interagir directamente com esse objecto de forma a alterar a sua geometria e/ou posição.

Aplicar indirectamente uma transformação a um objecto significa que estamos a interagir com um outro objecto qualquer, e por estes se encontrarem ligados, a geometria e/ou posição do primeiro é alterada.

Exemplo Visual



Requisitos Funcionais

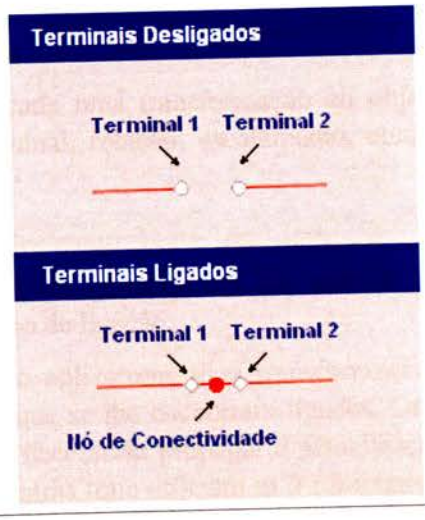
Nº	Descrição
1	A edição da topologia da rede é realizada através da interacção com as representações dos equipamentos da rede. Isto é, existem diagramas onde são representados estes equipamentos, e o utilizador usa o rato para mover ou de outra forma transformar estas representações de modo a estabelecer / desfazer ligações entre os terminais dos equipamentos representados.
2	Um utilizador pode mover: ▪ Uma colocação, ▪ Um barramento, ▪ Um equipamento de ligação, e ▪ Um gráfico de ligação, De forma a: ▪ Criar novas ligações (conectividade obtida por análise da proximidade / justaposição gráfica de terminais, operação esta que é feita no fim da translação), ▪ Manter as ligações existentes, e ▪ Remover as ligações existentes (estas são removidas no início da translação).
3	Um utilizador pode mover um vértice de um Barramento de forma a: ▪ Criar novas ligações (conectividade obtida por análise da proximidade / justaposição gráfica de terminais, operação esta que é feita no fim da translação), ▪ Manter as ligações existentes, e ▪ Remover as ligações existentes (estas são removidas no início da translação).

Nº	Descrição
4	Um utilizador pode mover um vértice de um: ▪ Equipamento de ligação, e ▪ Gráfico de ligação, Transformação esta que não tem nenhum impacto na conectividade.
5	Um utilizador pode mover um terminal de: ▪ Uma colocação, ▪ Um equipamento de ligação, e ▪ Um gráfico de ligação, De forma a: ▪ Criar novas ligações (conectividade obtida por análise da proximidade / justaposição gráfica de terminais, operação esta que é feita no fim da translação), ▪ Manter as ligações existentes, e ▪ Remover as ligações existentes (estas são removidas no início da translação).
6	Um utilizador pode reduzir ao mínimo o número de gráficos de ligação presentes num diagrama (algoritmo de redução dos gráficos de ligação).

Requisitos de Operação

Nº	Descrição
1	A selecção simples de um objecto é feita do seguinte modo: 1. Pressionar o botão esquerdo do rato sobre o objecto pretendido.
2	A selecção múltipla de objectos é feita dos seguintes modos: 1. Pressionar o botão esquerdo do rato sobre um ponto que não contenha nenhum objecto. 2. Arrastar o rato de forma a definir uma área de selecção. 3. Largar o botão esquerdo do rato. Ou 1. Pressionar a tecla *Shift*. 2. Pressionar o botão esquerdo do rato sobre o objecto pretendido. a. Este fica seleccionado caso não o esteja, ou b. Deixa de o estar em caso contrário. 3. Voltar ao ponto 2.

Nº	Descrição
3	A translação de um objecto é feita do seguinte modo: 1. Pressionar o botão esquerdo do rato sobre o objecto pretendido, mas não sobre um dos seus terminais e/ou vértices. ▪ Se a opção 'Remover Ligações' estiver activa, são removidas todas as ligações do objecto. 2. Arrastar o rato de forma a deslocar o objecto. ▪ A posição do objecto em questão é alterada (Transformação directa). ▪ A posição e/ou geometria dos objectos que a si estão ligados é alterada (Transformação indirecta). Ver **Actualização Indirecta dos Objectos**. ▪ É dada indicação gráfica de quais as ligações que poderemos estabelecer caso passarmos para o ponto 3. 3. Largar o botão esquerdo do rato. ▪ Conectividade obtida por análise da proximidade / justaposição gráfica de terminais.
4	A translação de um terminal de um objecto é feita do seguinte modo: 1. Pressionar o botão esquerdo do rato sobre o terminal pretendido. ▪ Se a opção 'Remover Ligações' estiver activa, são removidas todas as ligações do terminal. 2. Arrastar o rato de forma a deslocar o terminal. ▪ A geometria do objecto é alterada (Transformação directa). ▪ A posição e/ou geometria dos objectos que a si estão ligados é alterada (Transformação indirecta). Ver **Actualização Indirecta dos Objectos**. ▪ É dada indicação gráfica de quais as ligações que poderemos estabelecer caso passarmos para o ponto 3. 3. Largar o botão esquerdo do rato. ▪ Conectividade obtida por análise da proximidade / justaposição gráfica de terminais.

Nº	Descrição
5	Quando 2 ou mais terminais se encontram ligados, temos um nó de conectividade, nó este que pode ser transladado do seguinte modo: 1. Pressionar o botão esquerdo do rato sobre o nó de conectividade. 2. Arrastar o rato de forma a deslocar o nó de conectividade. ▪ A posição do nó de conectividade é alterada (Transformação directa). ▪ A posição e/ou geometria dos objectos que a si estão ligados é alterada (Transformação indirecta). Ver **Actualização Indirecta dos Objectos**. ▪ É dada indicação gráfica de quais as ligações que poderemos estabelecer caso passarmos para o ponto 3. 3. Largar o botão esquerdo do rato. ▪ Conectividade obtida por análise da proximidade / justaposição gráfica de terminais. Um nó de conectividade representa a ligação de vários terminais, e por isso não possibilita o acesso a cada um destes (sobreposição gráfica). Para que seja possível aceder a cada um desses terminais, *handles* especiais são disponibilizados: 
6	De forma a facilitar a interacção com o utilizador são disponibilizados diversos *Hotspots* que mais não são do que indicações gráficas. Estes são disponibilizados do seguinte modo: 1. Ao mover o rato, e tendo em conta uma área de aplicação, os *Hotspots* disponíveis são visualizados. Exemplo (s): ▪ Se o rato se encontrar "perto" de um terminal, este é visualizado. ▪ Se o rato se encontrar "perto" de um vértice, este é visualizado. ▪ Se o rato se encontrar "perto" de um nó de conectividade, os *handles* para os vários terminais a si ligados são visualizados.

Actualização Indirecta dos Objectos

Conforme foi mencionado, quando aplicamos uma transformação directamente a um objecto gráfico, a sua posição e/ou geometria é alterada. No entanto é necessário actualizar os objectos que se lhe encontram ligados de forma a manter as ligações correctas em termos visuais. Para tal definem-se 2 características que todos os objectos possuem:

- **Indirect Update Allowed**

  Indica se o objecto pode ou não sofrer uma actualização indirecta. Em caso de ser possível uma actualização indirecta então a actualização que lhe é aplicada é dada pelo Update Type (ver adiante). Em caso contrário, isto significa que o objecto não sofrerá nenhuma transformação indirecta, e por isso é necessário recorrer a uma gráfico de ligação para manter as ligações visualmente correctas. É este facto que justifica a necessidade dos gráficos de ligação.

- **Update Type**

  Existem 3 tipos de actualização distintos:

  - **Move Update**

    Neste caso é aplicada uma translação ao objecto.

  - **Transform Update**

    Neste caso é aplicada uma transformação ao objecto que não a translação (translação de terminal, rotação, escalamento, etc.) sendo esta específica de cada objecto.

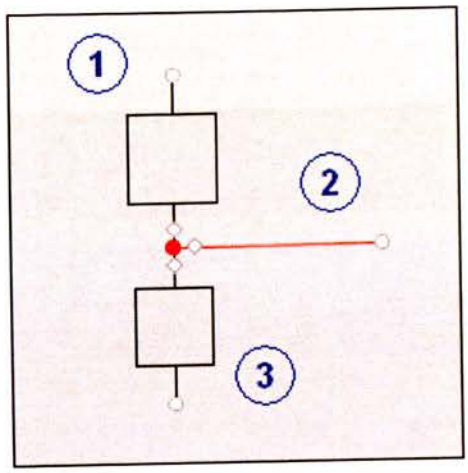
  - **Stand Still Update**

    Neste caso o objecto permanecerá na mesma posição sendo necessário adicionar um gráfico de ligação.

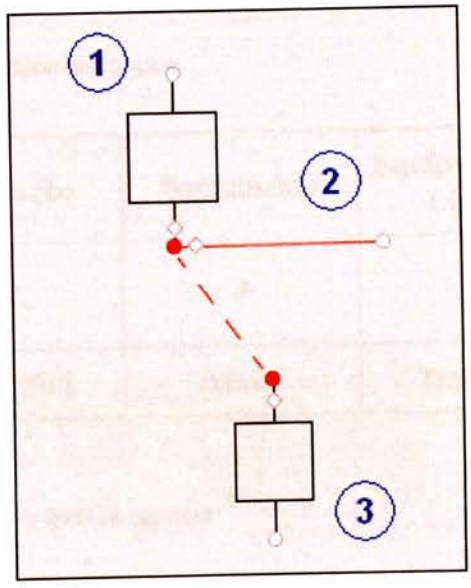
Esta actualização é recursiva. Ao aplicarmos uma transformação directa a um objecto, é necessário actualizar os objectos que se lhe encontram ligados. Ora, se estes alterarem a sua posição e/ou geometria, pode ser necessário propagar a actualização para outros objectos. A actualização global é regida por regras (que utilizam as 2 características mencionadas atrás) e termina quando todos os objectos e respectivas ligações estão visualmente correctos. Os detalhes deste algoritmo serão apresentados mais à frente.

Qual a diferença entre um objecto não permitir Indirect Update Allowed e permitir sendo a sua actualização do tipo Stand Still?

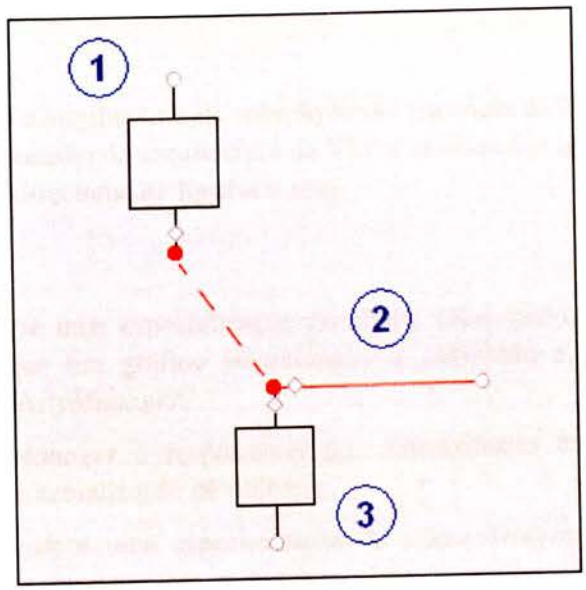
Consideremos a seguinte situação:



- Se o objecto 1 não permitir *Indirect Update Allowed* e o objecto 2 permitir (sendo a sua actualização do tipo *Move*), ao movermos o objecto 3 obtemos:



- Se o objecto 1 permitir *Indirect Update Allowed* (sendo a sua actualização do tipo *Stand Still*) e o objecto 2 permitir (sendo a sua actualização do tipo *Move*), ao movermos o objecto 3 obtemos:



Valores iniciais para os diferentes tipos de objectos

	Colocação	**Barramento**	**Equipamento de Ligação**	**Gráfico de Ligação**
Indirect Update Allowed	✗	✓	✓	✓
Update Type	Stand Still	Move	Transform	Transform

Valores possíveis para os diferentes tipos de objectos

	Colocação	**Barramento**	**Equipamento de Ligação**	**Gráfico de Ligação**
Indirect Update Allowed	✓✗	✓✗	✓✗	✓
Update Type	Move Transform Stand Still	Move Stand Still	Move Transform Stand Still	Transform

Os valores para o gráfico de ligação não podem ser alterados pois este tipo de objecto existe para resolver os casos em que não é possível actualizar indirectamente outros objectos. Por

isso mesmo, é sempre possível aplicar uma transformação indirecta a um gráfico de ligação, e esta será sempre do tipo Transform.

4.3 Arquitectura

Apresenta-se de seguida a arquitectura da solução tendo em conta as funcionalidades referidas anteriormente. A compreensão da arquitectura da VEF é essencial já que a maioria das classes aqui apresentadas estão directamente ligadas a esta.

diagram_editor

A classe *DiagramView* é uma especialização da classe *DiagramView* da VEF. Notifica o modelo de cada vez que um gráfico seleccionado é removido e, para além disso, está associada a um *ConnectivityManager*.

A classe *ConnectivityManager* é responsável por determinadas etapas e validações que ocorrem no algoritmo de actualização de objectos.

A classe *AbstractDiagram* é uma especialização da classe *Diagram* da VEF de forma a permitir que um diagrama pertença ao modelo. Isto é conseguido através da interface *RowInterface*. A classe *Diagram* é uma implementação concreta da classe *AbstractDiagram*.

A classe *Symbol* representa um gráfico composto por múltiplos gráficos (daí estenda a classe *GroupGraphic* da VEF). Um *Symbol* pode ser adicionado a um diagrama através de uma colocação. Uma colocação é um gráfico que está associado com um *Symbol*, sendo representada por este num determinado estado. Este estado é dado pelos valores actuais de um equipamento que também está associado à colocação.

A interface *ConnectivityGraphic* define métodos que todos os gráficos relacionados com conectividade devem implementar (colocação, barramento, equipamento de ligação, gráfico de ligação e nó de conectividade). A interface *EquipmentGraphic* é uma especialização da interface *ConnectivityGraphic* e deve ser implementada apenas por gráficos representando equipamentos (colocação, barramento, equipamento de ligação) e pelo gráfico de ligação (e por isso excluindo o nó de conectividade). Ambas as interfaces definem métodos comuns aos objectos relacionados com conectividade, estando estes relacionados com o algoritmo de actualização de objectos.

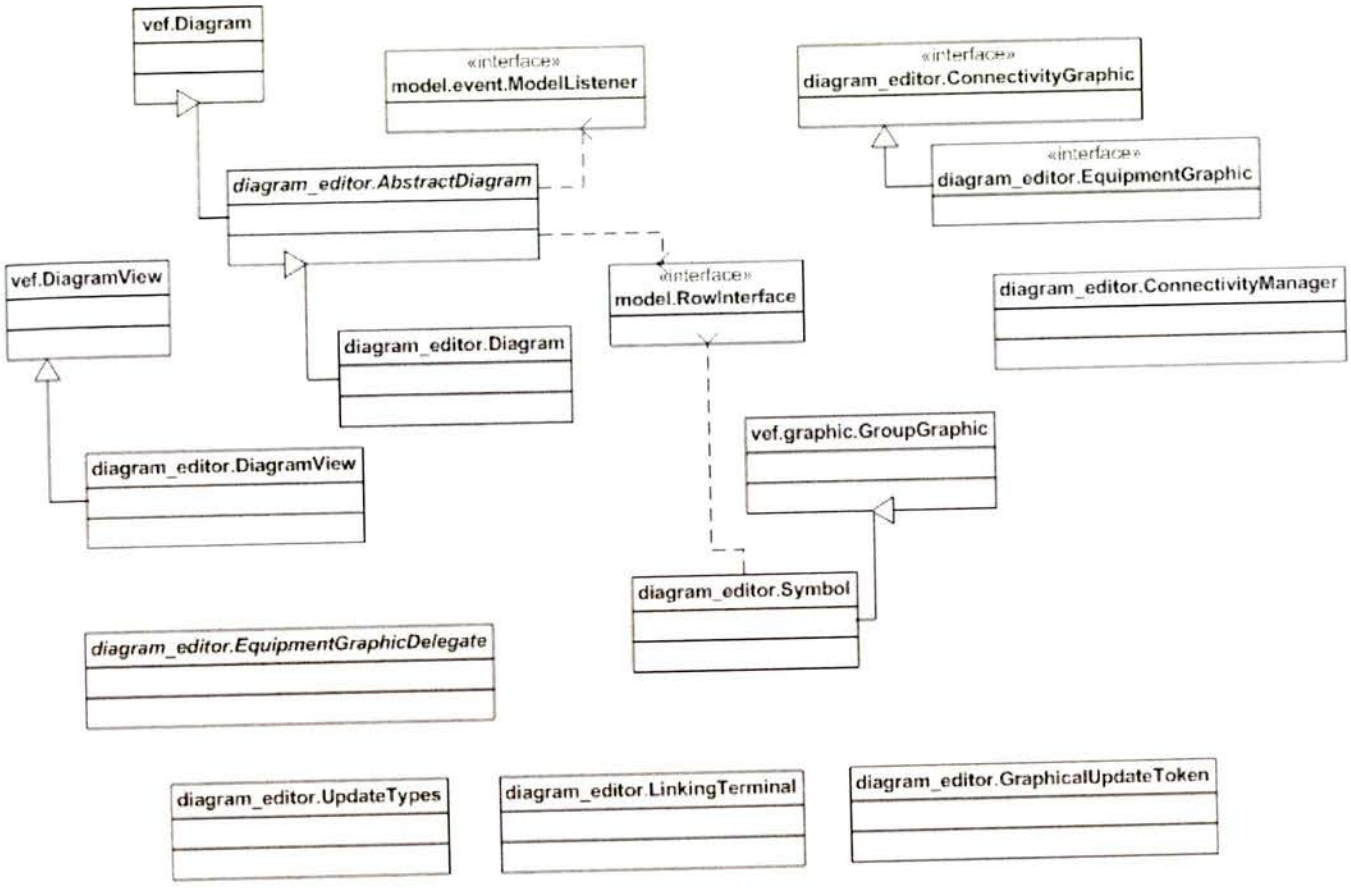
Tendo em conta que a linguagem Java não possibilita herança múltipla e também o facto de que existem inúmeras funcionalidades semelhantes a todos os objectos de conectividade, a classe *EquipmentGraphicDelegate* existe de forma a fornecer uma implementação comum, sem necessidade de haver repetição de código.

A classe *GraphicalUpdateToken* representa um *token* que é passado aos objectos no algoritmo de actualização de objectos. Entre outras funcionalidades, este *token* permite identificar ciclos no algoritmo.

A classe *UpdateTypes* define os diferentes tipos de actualização referidos anteriormente.

A classe *LinkingTerminal* representa um terminal de ligação. Os objectos gráficos têm um ou mais terminais de ligação.

Diagrama de Classes



diagram_editor.graphic

As classes *TerminalsPlacement*, *BUS*, *LinkingEquipment* e *LinkingGraphic* representam respectivamente: Colocação, Barramento, Equipamento de Ligação e Gráfico de Ligação. Conforme foi referido, todas elas implementam a interface *EquipmentGraphic*. As classes *BUS*, *LinkingEquipment* e *LinkingGraphic* estendem todas elas a classe *Polyline*, classe esta que é uma especialização da classe *Polyline* da VEF (difere no sentido de que necessita de ser consistente e por isso implementa a interface *RowInterface*).

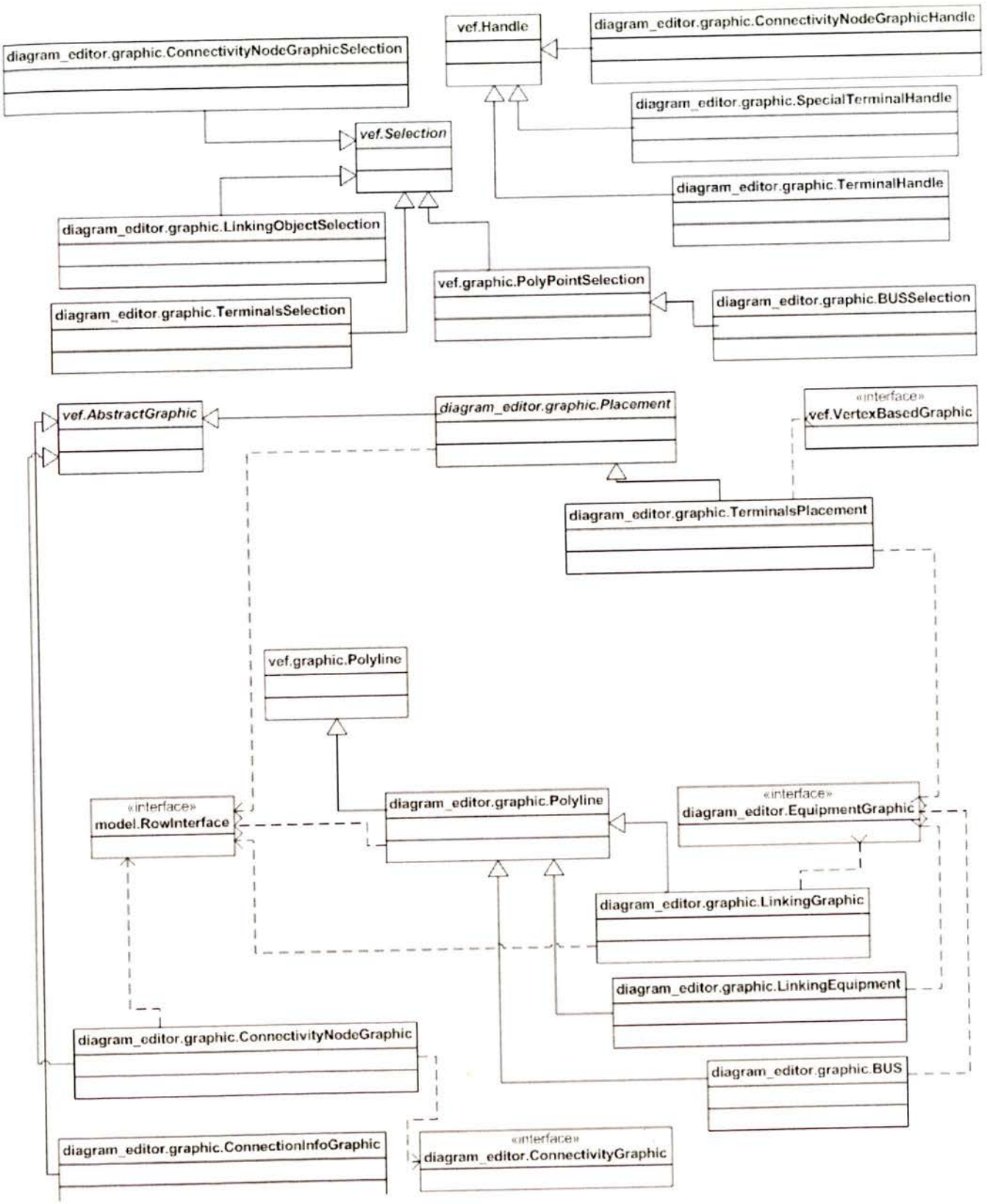
A classe *ConnectivityNodeGraphic* representa um nó de conectividade e como foi referido implementa a interface *ConnectivityGraphic*.

A classe *ConnectionInfoGraphic* representa o gráfico que é mostrado ao utilizador para este ter noção de que ligações podem ser estabelecidas, passo este que pertence ao algoritmo de actualização de objectos.

De seguida temos as selecções dos diferentes tipos de objectos. A classe *TerminalsSelection* representa a selecção de uma colocação; A classe *BUSSelection* representa a selecção de um barramento; A classe *LinkingObjectSelection* representa a selecção de um equipamento de ligação e de um gráfico de ligação; e a classe *ConnectivityNodeGraphicSelection* representa a selecção de um nó de conectividade. Todas elas estendem a classe *Selection* da VEF.

Finalmente temos os novos *handles*. Temos *handles* de terminais (*TerminalHandle*), *handles* especiais aquando da presença de um nó de conectividade (*SpecialTerminalHandle*) e os *handles* do próprio nó de conectividade (*ConnectivityNodeGraphicHandle*).

Diagrama de Classes



diagram_editor.interactor

Na *package diagram_editor.interactor* existem definidas várias subclasses de *Interactor*, que podemos novamente classificar em interactores de vista, interactores de criação de gráficos e interactores de modificação de gráficos.

Interactores de Vista

A classe *SelectionInteractor* estende a classe *SelectionInteractor* da VEF pois, além da funcionalidade já mencionada, permite a visualização de *Hotspots* (ao mover o rato).

Interactores de Criação de Gráficos

Foram criadas diversas classes que permitem a criação dos novos tipos de gráficos. Todas elas estendem classes da VEF, acrescentando unicamente a funcionalidade de registo no modelo dos gráficos criados. Devido à simplicidade deste facto estas classes não são mostradas no diagrama de classes.

Interactores de Modificação de Gráficos

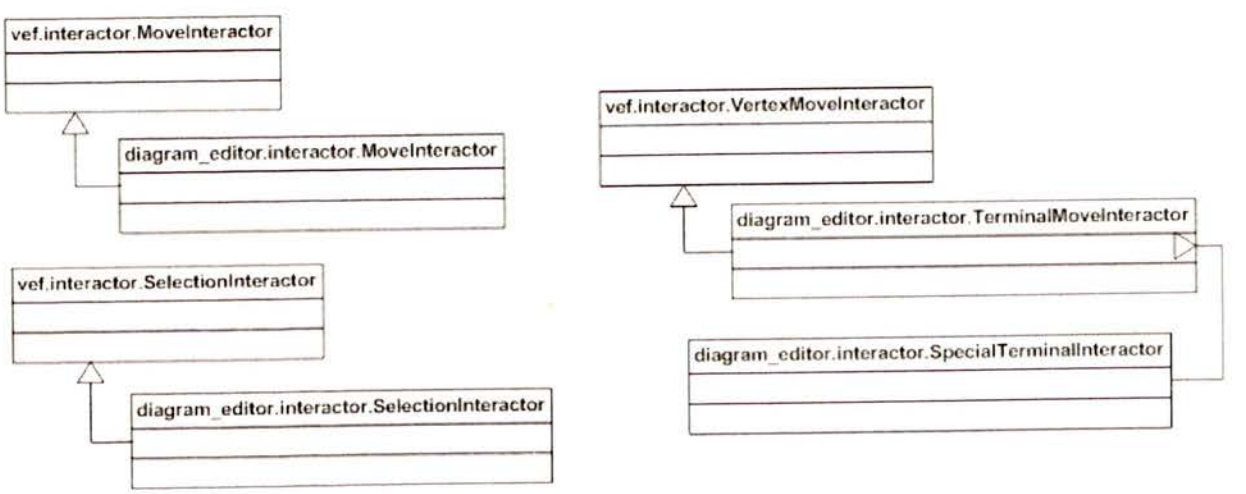
Todos os interactores da VEF que modificam a posição e/ou geometria dos objectos gráficos necessitavam de ser estendidos. Isto porque quando aplicamos uma transformação directamente a um objecto é necessário actualizar aqueles que lhe estão ligados.

A classe *MoveInteractor* estende a classe *MoveInteractor* da VEF e possui toda a funcionalidade desta com o acrescento de que corre o algoritmo de actualização dos objectos.

Da mesma forma, a classe *TerminalMoveInteractor* estende a classe *VertexMoveInteractor* da VEF e possui toda a funcionalidade desta com o acrescento de que corre o algoritmo de actualização dos objectos.

A classe *SpecialTerminalInteractor* tem o mesmo comportamento da classe *TerminalMoveInteractor*, excepção feita ao facto de que no início da interacção o rato é deslocado automaticamente do *handle* especial para a posição real do terminal de ligação.

Diagrama de Classes



4.4 Implementação

De entre os muitos pormenores da implementação, e por uma questão de simplicidade, nesta secção apresentam-se apenas os mais relevantes. Todos eles estão ligados com o algoritmo de actualização de objectos.

Algoritmo de actualização indirecta dos objectos

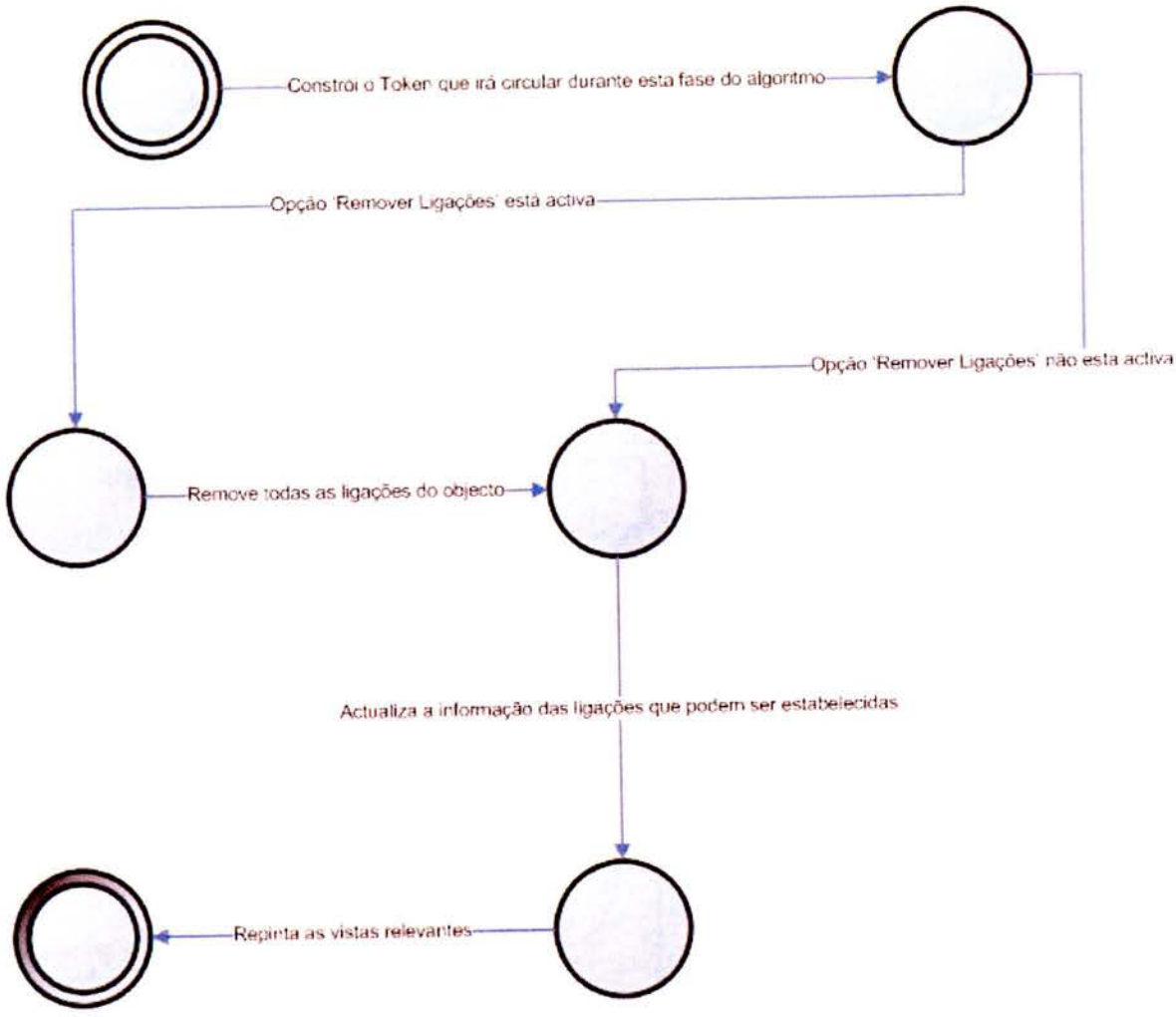
Recorrendo a diagramas de estado para melhor compreensão, apresenta-se de seguida o algoritmo de actualização dos objectos. Mas antes, explicam-se alguns conceitos.

O algoritmo pode ser divido em 3 fases distintas. Estas são o Início, Durante e Fim correspondendo respectivamente a *MousePressed*, *MouseDragged* e *MouseReleased*, etapas estas que representam o funcionamento dos interactores relacionados com conectividade, definidos anteriormente.

A **regra básica** do algoritmo é que a localização de um terminal de ligação deve sempre coincidir com a localização do nó de conectividade ao qual está ligado (caso esteja ligado a algum).

Início

O algoritmo começa no objecto sobre o qual estamos a aplicar uma transformação directa. O interactor em questão passa o controlo para esse mesmo objecto (através da interface *ConnectivityGraphic* – método *graphicalUpdateStarted*). O objecto gráfico constrói o token desta fase, remove as suas ligações caso essa opção esteja activa (informação dada pelo *ConnectivityManager*), actualiza a informação das ligações que podem ser estabelecidas (as ligações que podem ser estabelecidas são dadas pelo *ConnectivityManager*) e finalmente repinta as vistas relevantes.



Durante

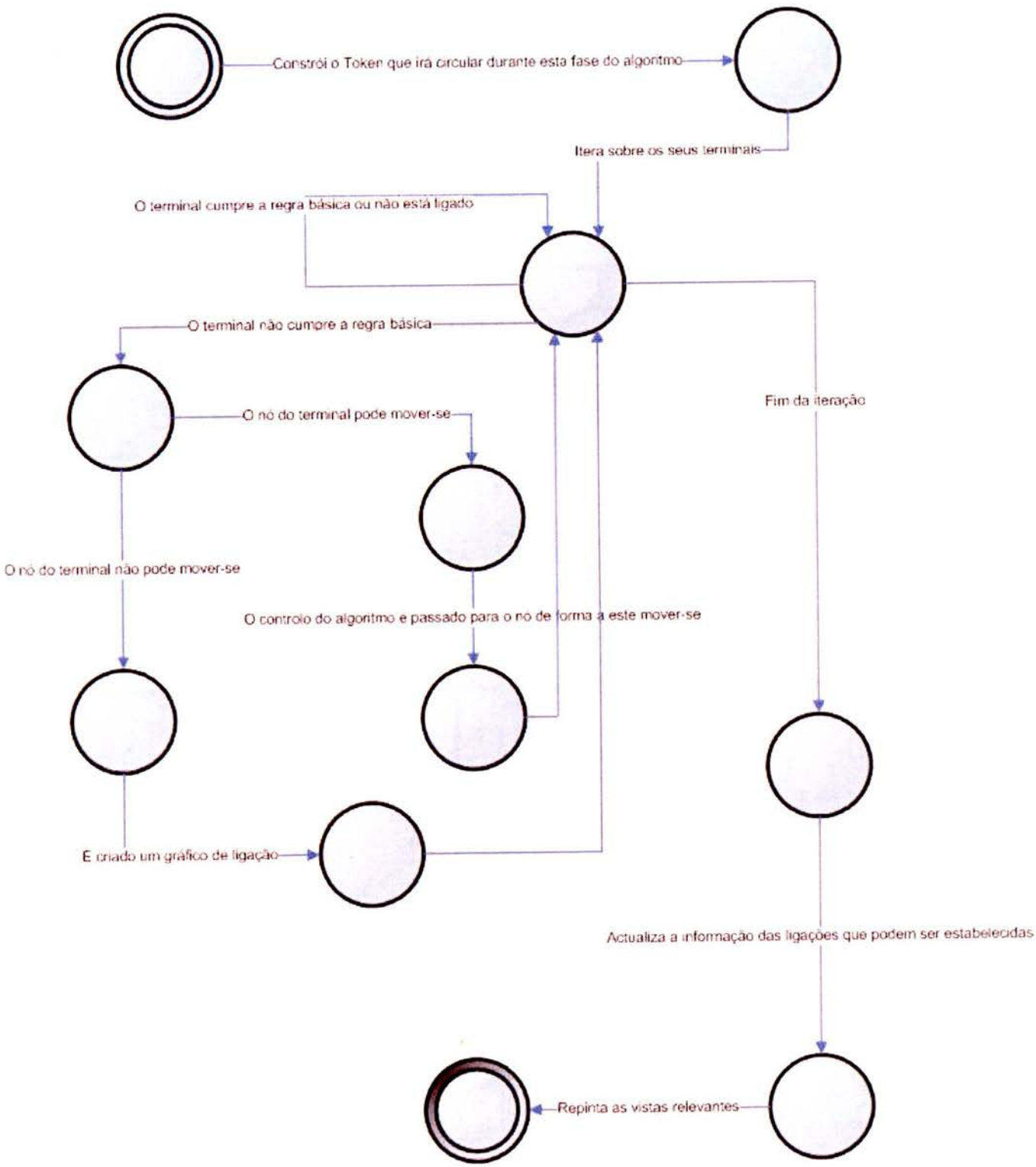
O interactor em questão passa o controlo para o objecto (através da interface *ConnectivityGraphic* – método *graphicalUpdate*).

Esta fase representa a fase de propagação sendo por isso a mais complexa. O objecto começa por construir o token que irá circular durante esta fase. De seguida itera sobre todos os seus terminais. Se o terminal cumpre a regra básica ou não está ligado então é ignorado e passa-se para o próximo. Se o terminal não cumpre a regra básica então é necessário propagar. Se o nó de conectividade ao qual este terminal está ligado não pode mover-se (explicado já de seguida) então é criado um gráfico de ligação de forma à regra básica passar a ser cumprida. Em caso contrário o controlo do algoritmo é passado para o nó pedindo-se a este que se mova para a localização do terminal em questão (explicado mais à frente). Por fim, o objecto gráfico actualiza a informação das ligações que podem ser estabelecidas (as ligações que podem ser estabelecidas são dadas pelo *ConnectivityManager*) e finalmente repinta as vistas relevantes.

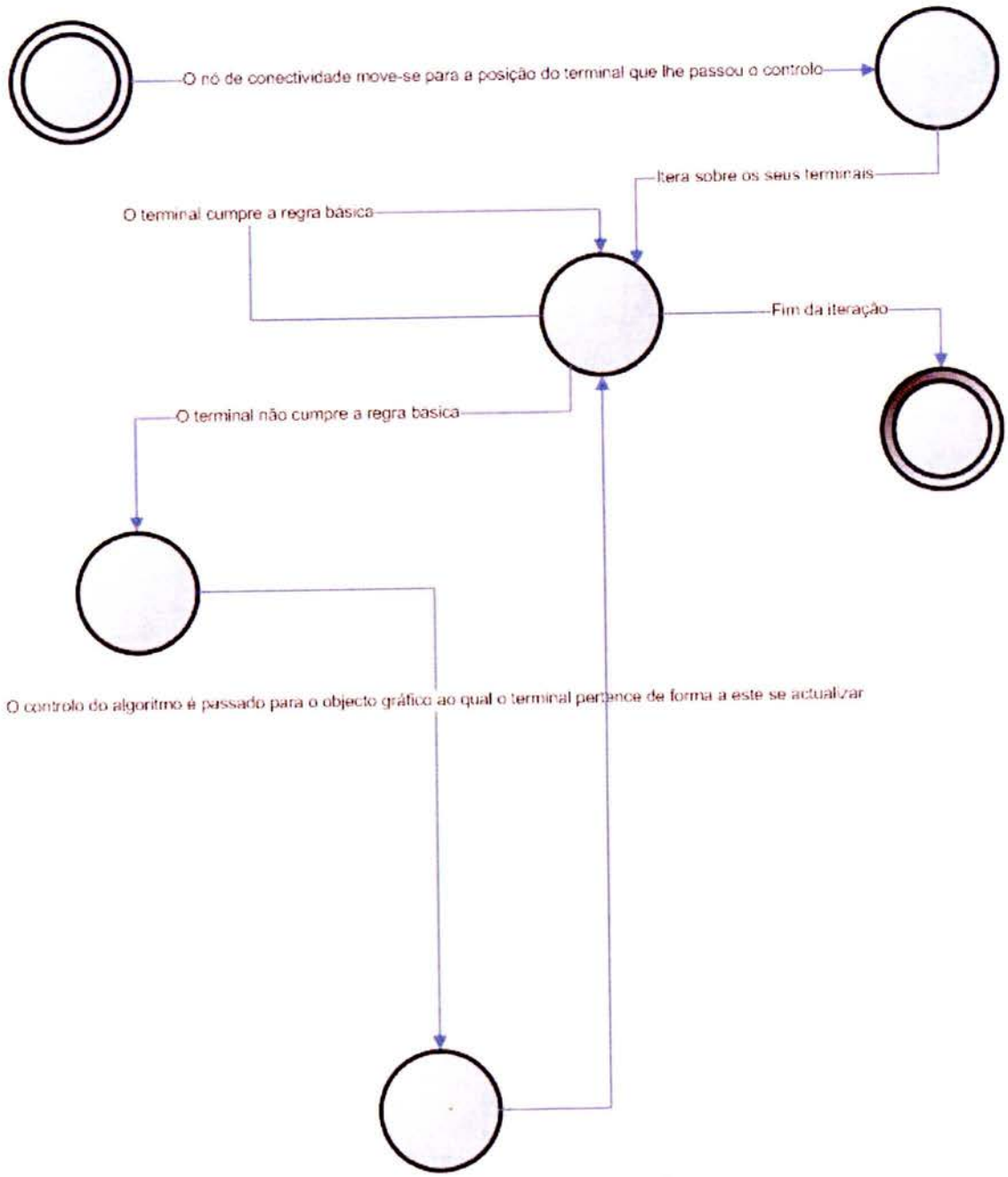
Um nó de conectividade pode mover-se se:

- Ainda não desempenhou nenhum papel no algoritmo (verificado através do *token* para evitar ciclos); e

- Todos os objectos gráficos (excepção feita ao que lhe passou o controlo) permitem actualização indirecta.

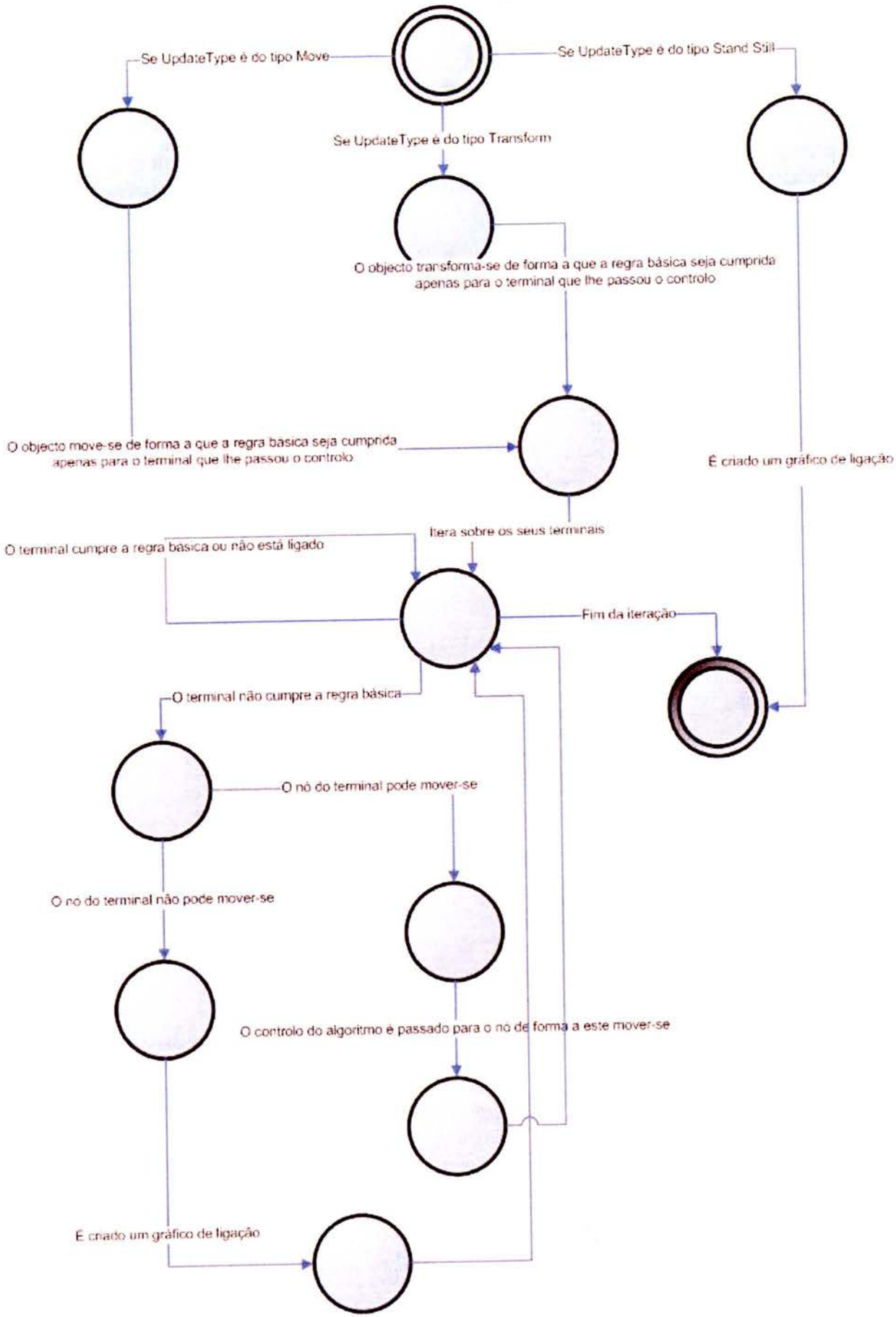


Agora que o controlo está no nó de conectividade, este move-se para a localização do terminal que lhe passou o controlo. De seguida itera sobre os seus terminais. Se o terminal cumpre a regra básica então é ignorado e passa-se para o próximo (o único nesta situação é o que passou o controlo). Se o terminal não cumpre a regra básica (os restantes) então é necessário propagar novamente. O controlo é então passado para o equipamento gráfico ao qual pertence o terminal, de forma a este se actualizar (explicado mais à frente).



O controlo está neste momento no equipamento gráfico. Este necessita de se actualizar. Se o seu tipo de actualização for *Move* ou *Transform* então o objecto move-se ou transforma-se para que a regra básica seja cumprida apenas para o terminal que lhe passou o controlo. Em ambos os casos tal actualização deve ser propagada. Sendo assim o objecto itera sobre todos os seus terminais. Se o terminal cumpre a regra básica ou não está ligado então é ignorado e passa-se para o próximo. Se o terminal não cumpre a regra básica então é necessário propagar (processo explicado anteriormente). No caso do seu tipo de actualização ser *Stand Still* então é criado um gráfico de ligação para que a regra básica seja cumprida.

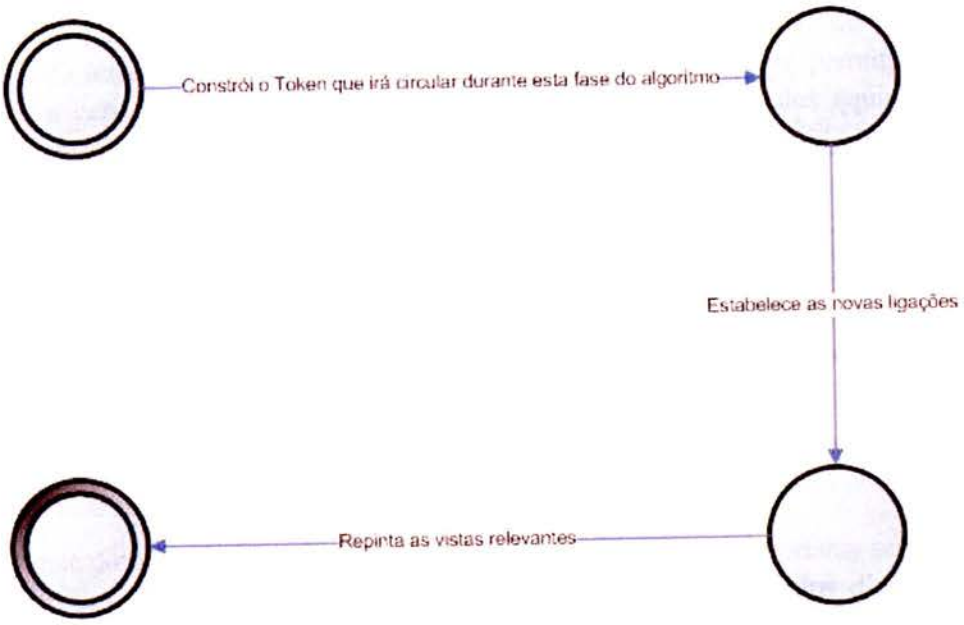
Como podemos constatar, o algoritmo propaga até que todos os terminais cumpram a regra básica, altura em que termina.


Se UpdateType é do tipo Move
Se UpdateType é do tipo Stand Still
Se UpdateType é do tipo Transform
O objecto transforma-se de forma a que a regra básica seja cumprida apenas para o terminal que lhe passou o controlo
O objecto move-se de forma a que a regra básica seja cumprida apenas para o terminal que lhe passou o controlo
É criado um gráfico de ligação
Itera sobre os seus terminais
O terminal cumpre a regra básica ou não está ligado
Fim da iteração
O terminal não cumpre a regra básica
O nó do terminal pode mover-se
O nó do terminal não pode mover-se
O controlo do algoritmo é passado para o nó de forma a este mover-se
É criado um gráfico de ligação

Fim

O interactor em questão passa o controlo para o objecto (através da interface *ConnectivityGraphic* – método *graphicalUpdateFinished*)

O objecto gráfico constrói o token desta fase, estabelece as novas ligações (as ligações que podem ser estabelecidas são dadas pelo *ConnectivityManager*) e finalmente repinta as vistas relevantes.



5 Conclusões

Apresentação do estado actual do projecto, trabalhos futuros e apreciação global do estágio.

5.1 Estado actual do projecto

Todas as funcionalidades referidas neste relatório foram implementadas, excepção feita ao algoritmo de redução de gráficos de ligação - este foi apenas pensado e tido em conta no desenvolvimento da arquitectura da solução. Sendo assim, no final do estágio, ficou desenvolvida uma solução estável e que cumpre os requisitos propostos, permitindo assim ao utilizador a captura gráfica da conectividade por edição/população dos equipamentos nos diagramas da rede eléctrica.

5.2 Trabalhos futuros

Como foi referido, o editor gráfico da rede eléctrica é uma parte integrante do sistema SCADA/DMS da EFACEC. Este sistema está neste momento a ser desenvolvido, e por isso, será necessário implementar novas funcionalidades, melhorar outras, efectuar testes de integração, testes de desempenho, etc.

No entanto, e de uma forma específica, as tarefas que logicamente seguem este estágio são:

- **Melhorar o desempenho do editor gráfico**

  Neste domínio podem ser feitas diversas coisas mas a mais importante será optimizar a estrutura de dados que serve de suporte ao armazenamento dos diversos gráficos presentes num diagrama. A visualização de *hotspots*, a selecção e até o próprio algoritmo de actualização necessitam de pesquisar objectos num determinado espaço. Ao optimizar a estrutura de dados, melhoramos os tempos de pesquisa e consequentemente melhoramos todo desempenho do editor.

- **Definir regras flexíveis de validação de conectividade**

  Esta questão, embora não tenha sido implementada, foi incorporada na arquitectura da solução, sendo por isso fácil de atingir. A definição de regras flexíveis de validação de conectividade permite, por exemplo, impedir que equipamentos de baixa tensão sejam capturados como estando ligados a equipamentos de alta tensão. É uma validação extra do algoritmo de actualização que diz o que se pode ligar e o que se pode desligar.

5.3 Apreciação global do estágio

A fase inicial do estágio foi algo complexa, tendo em conta que a solução pretendida inseria-se num projecto em desenvolvimento de considerável dimensão. Foi necessária uma análise cuidada das tecnologias e aplicações existentes, bem como, do trabalho desenvolvido por um colega de estágio que já havia estudado alguns pontos em comum. Não obstante, os objectivos do estágio foram cumpridos, resultando deste uma solução que cumpre os requisitos fundamentais respeitantes à conectividade.

A instituição, através da sua infra-estrutura e política de funcionamento, possibilita a todos os seus funcionários excelentes condições de trabalho e também de lazer. O mesmo se pode dizer

de todas as pessoas com quem trabalhei. Funcionam como uma equipa o que leva a que haja um bom ambiente de trabalho.

6 Bibliografia

[1] EFACEC, http://www.eface.pt/, Setembro de 2005.

[2] Sun Developer Network (SDN), http://java.sun.com/, Setembro de 2005.

[3] Pedro Silva, Arrasto de objectos, EFACEC SE, Abril de 2004.

[4] Pedro Silva, Requisitos do Protótipo do Editor Gráfico – Conectividade, EFACEC SE, Abril de 2004.

[5] Pedro Silva, Teste Unitário Protótipo Editor Gráfico – Conectividade, EFACEC SE, Abril de 2004.

[6] José Henriques, Desenho Protótipo Editor Gráfico, EFACEC SE, Abril de 2004.

[7] Dave Marsh, Sumário do Ambiente dos Editores, EFACEC SE, Fevereiro de 2005.

[8] Pedro Silva, Desenho Framework Edição Gráfica, EFACEC SE, Maio de 2005.

[9] João Mendes, Especificação do Editor Gráfico da Rede Eléctrica – Conectividade, EFACEC SE, Outubro de 2005.

[10] João Mendes, Desenho do Editor Gráfico da Rede Eléctrica – Conectividade, EFACEC SE, Novembro de 2005.

Faculdade de Engenharia da Universidade do Porto
Rua Dr. Roberto Frias, s/n 4200-465 Porto PORTUGAL
www.fe.up.pt