

Faculdade de Engenharia da Universidade do Porto
Licenciatura em Engenharia Informática e Computação



m-GIS: *Mobile GIS*

INESC Porto

Relatório do Estágio Curricular da LEIC 2002/2003

Jorge Carlos dos Santos Cardoso

Orientador na FEUP: Prof. João Correia Lopes

Orientador no INESC Porto: Eng. Artur Rocha

Setembro de 2003

Resumo

O objectivo do estágio *m-GIS* — *mobile Geographic Information System* é desenvolver um sistema que permita a visualização de informação geográfica em dispositivos móveis, nomeadamente telemóveis e *PDA*s. A informação geográfica está representada no formato *Geography Markup Language* (GML) e localizada num servidor. Uma vez que se pretende que o sistema seja multi-plataforma, deverá ser usada a tecnologia Java.

O objectivo deste estágio deve ser encarado numa perspectiva de prova de conceito. Quer isto dizer que a finalidade do projecto é demonstrar que é possível implementar um sistema que cumpra os requisitos apresentados. Por outro lado, pretende-se também compreender as limitações do sistema no contexto das tecnologias actualmente em uso e perceber de que forma tecnologias futuras poderão ser usadas para as ultrapassar.

O trabalho desenvolvido no âmbito deste projecto consistiu em duas fases distintas. Primeiro foi desenvolvido um sistema, em que a informação geográfica se encontrava num servidor em ficheiros GML, que correspondeu ao objectivo traçado no início do projecto. Numa segunda fase, e uma vez que o objectivo primário havia sido atingido, foi desenvolvido um sistema que consistiu na integração do anterior com um *Web Feature Server* (WFS), que fornecia os dados geográficos.

O sistema foi desenvolvido usando uma arquitectura cliente/servidor tendo por base a tecnologia Java. No caso do cliente, a tecnologia Java usada foi o J2ME (*Java 2 Micro Edition*).

No primeiro protótipo *m-GIS* a informação geográfica encontrava-se armazenada em ficheiros GML, sendo transformada (usando *Extensible Stylesheet Language Transformations* — XSLT) em *Scalable Vector Graphics* (SVG), em tempo de execução, e enviada para o cliente. No segundo, a informação geográfica em formato GML era servida por um WFS, sendo transformada em SVG e enviada ao cliente. Os pedidos ao servidor, em ambos os casos, são feitos através de HTTP.

Para se poder visualizar documentos SVG no cliente foi necessário implementar um interpretador de SVG (*parser* de SVG) e um visualizador de SVG.

O objectivo principal deste projecto era obter uma resposta à pergunta: “será viável desenvolver um sistema móvel de visualização de informação geográfica recorrendo a normas e formatos abertos”? Os resultados obtidos indicam que sim, mas as severas limitações de recursos e de poder de processamento nos dispositivos alvo, impuseram também limitações ao sistema desenvolvido: limitações ao nível da quantidade de informação que pode ser visualizada pelo utilizador, limitações ao nível da velocidade com que a informação é processada pelo cliente e limitações gráficas dos mapas gerados.

Agradecimentos

O trabalho desenvolvido neste estágio não teria sido possível sem a ajuda de várias pessoas, a quem quero agradecer.

Quero, em primeiro lugar, agradecer ao Engenheiro Artur Rocha, o meu orientador no INESC Porto, pelo acompanhamento e apoio prestados ao longo do trabalho.

Desejo agradecer a toda a equipa de GIS do INESC Porto, pela sua disponibilidade e empenho no projecto de estágio. Quero agradecer em especial ao Rui Chilro, Luís Bártolo, Marco Amaro, José Lino e Lígia Silva.

Ao meu orientador na FEUP, o Professor João Correia Lopes, pelo interesse e disponibilidade demonstrados ao longo deste projecto e por todo o apoio prestado.

À minha família pelo apoio incondicional.

A todos os outros que me esqueci de mencionar.

Finalmente, quero agradecer o financiamento do PRODEP.

Conteúdo

1	Introdução	1
1.1	Apresentação do INESC Porto	1
1.2	O Projecto m-GIS no INESC Porto	2
1.3	Organização e Temas Abordados	3
2	Sistemas de Informação Geográfica e a Arena Móvel	5
2.1	Sistemas de Informação Geográfica	5
2.2	<i>Geography Markup Language</i> (GML)	6
2.3	<i>Scalable Vector Graphics</i> (SVG)	7
2.4	O <i>Java Micro Edition</i> (J2ME)	7
2.4.1	Visão Geral	8
2.4.2	<i>J2ME Wireless Toolkit</i> (WTK)	9
2.4.3	Algumas Notas Sobre J2ME e Dispositivos Reais	9
2.4.4	MIDP e SVG	10
2.4.5	MIDP e XML	10
2.4.6	MIDP e <i>Web Services</i>	10
2.4.7	MIDP e Números de Vírgula Fixa	11
2.5	<i>Web Feature Services</i> e <i>Web Map Services</i>	11
2.5.1	<i>Web Map Services</i> (WMS)	11
2.5.2	<i>Web Feature Services</i> (WFS)	12
2.5.3	<i>deegree</i>	13
2.6	Sistemas Existentes	13
3	O Sistema m-GIS	15
3.1	Visão Geral	15
3.2	Arquitectura Proposta	16
3.3	Tecnologias	18

3.4	O Servidor m-GIS	19
3.5	O Cliente	19
3.6	Integração com um WFS	21
4	Implementação do Sistema m-GIS	24
4.1	Ambiente de Programação	24
4.2	O Servidor m-GIS	24
4.3	O Cliente	28
4.3.1	Interacção	30
4.4	O Visualizador de SVG	31
4.5	Optimizações	35
4.5.1	Interpretação de SVG	35
4.5.2	Transformação GML para SVG	38
4.6	Integração com um WFS	40
4.6.1	O Servidor	40
4.6.2	O cliente	42
5	Exploração e Testes	44
5.1	O Sistema m-GIS em Acção	44
5.2	Limites do Sistema m-GIS	46
5.2.1	Análise dos Resultados	49
6	Conclusões e Trabalho Futuro	50
6.1	Melhorias ao Sistema Desenvolvido	50
6.2	Novas Funcionalidades	51
	Glossário	56
A	J2ME	59
A.1	Introdução	59
A.2	<i>Connected Limited Device Configuration (CLDC)</i>	60
A.2.1	Requisitos Mínimos	60
A.2.2	Diferenças da CLDC para um ambiente Java “normal”	61
A.3	<i>Mobile Information Device Profile (MIDP)</i>	64
A.3.1	Bibliotecas do MIDP	65
A.3.2	<i>MIDlets</i>	66

A.4	Pacotes opcionais	66
B	GML	68
B.1	Visão Geral	68
B.2	Modelo de Objectos	69
B.3	Codificação de GML	70
C	SVG	75
C.1	<i>Rendering</i>	75
C.2	Estrutura do Documento SVG	76
C.3	Caminhos (<i>Paths</i>)	76
C.4	Elementos Básicos	77
C.5	Transformações	78
C.6	SVG Tiny	80

Lista de Figuras

2.4.1	Arquitectura de um ambiente J2ME	8
2.5.1	Arquitectura geral de um WFS	13
3.1.1	Arquitectura de alto nível do sistema m-GIS	16
3.2.1	Arquitectura proposta para o sistema m-GIS	16
3.2.2	Diagrama dos ecrãs da aplicação do cliente	17
3.5.1	Arquitectura da aplicação do cliente	20
3.6.1	Arquitectura geral do sistema m-GIS com integração do WFS	22
4.2.1	Diagrama de classes do servidor	25
4.3.1	Diagrama das principais classes da aplicação cliente	30
4.4.1	Diagrama de classes da estrutura do visualizador SVG	32
4.4.2	A árvore de objectos correspondente ao Exemplo 4.4.1	34
4.4.3	A classe <i>Matrix</i>	35
4.5.1	Diagrama de classes alteradas no kXML	37
4.6.1	Diagrama de classes do servidor m-GIS na nova configuração	41
4.6.2	Diferença entre a área pedida e a área recebida usando um WFS	42
A.1.1	Arquitectura de um ambiente J2ME	59
A.3.1	Arquitectura de um dispositivo MIDP (adaptado de [Sun00b])	64
B.3.1	Esquema geométrico (retirado de [OCG01])	71
B.3.2	Esquema de objectos geográficos (retirado de [OCG01])	72
C.4.1	Aspecto gráfico do Exemplo C.4.1	79

Lista de Exemplos

4.2.1 Exemplo de um documento GML	27
4.2.2 Resultado da transformação do documento do Exemplo 4.2.1	28
4.4.1 Um exemplo de SVG	33
4.5.1 Parte de um documento SVG típico	36
4.5.2 Parte de um documento SVG típico usando o elemento <code><path></code>	39
C.2.1 Exemplo de um documento SVG	76
C.3.1 Exemplo de um caminho	77
C.4.1 Exemplo de formas básicas em SVG	79

Lista de Tabelas

2.4.1 Interpretadores XML para J2ME	10
4.5.1 Desempenho da interpretação de documentos SVG.	38
4.5.2 Tamanho do documento resultante das transformações, antes e depois da optimização	39
4.5.3 Tempo de transformação dos ficheiros antes e depois da optimização	39
4.5.4 Tempo de interpretação dos ficheiros antes e depois da optimização.	40
5.2.1 Dimensões máximas dos mapas de acordo com a memória disponível	47
5.2.2 Tempos de transformação e de interpretação	48
5.2.3 Tempos de de interpretação num dispositivo real: HP Jornada	48
B.2.1 Propriedades geométricas básicas (retirado de [OCG01])	70
C.3.1 Instruções do elemento <code><path></code>	77

Capítulo 1

Introdução

Neste capítulo é apresentada a instituição onde decorreu o projecto de estágio, são apresentados os objectivos e enquadramento do estágio na instituição e, finalmente, é descrita a organização do presente documento.

1.1 Apresentação do INESC Porto

O INESC Porto — Instituto de Engenharia de Sistemas e Computadores do Porto é uma associação privada sem fins lucrativos, declarada de utilidade pública, constituída em Dezembro de 1998, cujos associados são o INESC, a Universidade do Porto e a Faculdade de Engenharia da Universidade do Porto. A sua criação foi o corolário de um processo de profunda reestruturação do INESC, que começou pela criação das condições para a especialização local dos vários pólos e pela sua autonomização e que conduziu à criação de um conjunto de instituições autónomas, ligadas centralmente ao INESC que constituirá um centro de orientação estratégica e consolidação nacional.

O INESC é, por sua vez, uma associação privada sem fins lucrativos e de utilidade pública, fundado em 1980 e que tem por associados empresariais a Portugal Telecom, a CPRM e os Correios de Portugal e como associados universitários o Instituto Superior Técnico, a Universidade Técnica de Lisboa, a Universidade do Porto, a Universidade de Aveiro e a Universidade de Coimbra.

Objectivos e Organização do INESC Porto

O INESC Porto é uma instituição criada para constituir uma interface entre o mundo académico e o sector das Tecnologias de Informação, Telecomunicações e Electrónica, dedicando-se à investigação e desenvolvimento científicos, transferência de tecnologia e formação avançada.

Constituem objectivos últimos do INESC Porto a produção de ciência e de tecnologia capaz de competir nos mercados nacionais e mundiais e a formação de recursos humanos de qualidade científica e técnica, motivados para apostar nas capacidades nacionais e na modernização do País.

A ligação ao ensino superior, a montante, que é assegurada pelo envolvimento de um número significativo de docentes no INESC Porto, constitui um contributo para a evolução do sistema

de ensino científico e tecnológico, modernizando-o e adaptando-o às necessidades do tecido económico e social.

Deste modo, o INESC Porto é, antes de mais, um instrumento estratégico do sector universitário português e, consciente do valor dos recursos humanos e do espaço de autonomia de que dispõe, dá, através da formação em serviço de jovens recém licenciados, da transferência de conhecimento e produtos para a indústria, o seu contributo para a construção de um Portugal moderno, de uma economia sólida e de uma sociedade de qualidade. Neste contexto, quer a internacionalização quer a escolha criteriosa de parcerias estratégicas são aspectos chave.

A articulação entre as actividades de I&D, transferência tecnológica e consultoria, realiza-se no âmbito das cinco Unidades em que está organizado:

- Unidade de Engenharia de Sistemas de Produção
- Unidade de Optoelectrónica e Sistemas Electrónicos
- Unidade de Sistemas de Energia
- Unidade de Telecomunicações e Multimédia
- Unidade de Sistemas de Informação e Comunicação

O acompanhamento, orientação e avaliação internos das actividades de carácter científico e técnico, competem a um Conselho Científico, existindo ainda uma Comissão de Acompanhamento Científico, constituída por personalidades do mundo científico, cobrindo as áreas de intervenção do INESC Porto e maioritariamente de origem estrangeira. Ambos estes órgãos estão consagrados como órgãos associativos, nos Estatutos do INESC Porto.

A instituição como um todo, as suas Unidades e Centro contam com o apoio de serviços administrativos adequados — o Departamento de Informação e Logística — DIL, bem como de um Departamento de Comunicações e Informática.

Unidade de Sistemas de Informação e Comunicação (USIC)

A Unidade de Sistemas de Informação e Comunicação estuda, desenvolve e promove soluções integradas no campo dos sistemas de informação e comunicação. A Unidade realiza diversos tipos de actividades, nomeadamente: desenvolvimento, transferência de tecnologia, consultoria, auditoria e formação. Estas actividades decorrem em vários sectores, salientando-se as Telecomunicações, Autarquias, Indústria, Comércio, Saúde e Administração Central e Regional. Em termos de competências chave, a Unidade cobre de uma forma geral os campos das telecomunicações e sistemas de informação, suportada por uma equipa heterogénea, com formação em áreas diversificadas.

1.2 O Projecto m-GIS no INESC Porto

O objectivo do estágio *m-GIS* — *mobile Geographic Information System* é desenvolver um sistema que permita a visualização de informação geográfica em dispositivos móveis, nomeadamente telemóveis e *PDA*s.

A informação geográfica deve estar representada no formato GML (*Geography Markup Language*) e localizada num servidor. Uma vez que se pretende que o sistema seja multi-plataforma a tecnologia a usar deverá ser Java.

O sistema deverá usar, na medida do possível, normas e formatos abertos. O objectivo é tornar o sistema completamente independente de tecnologias proprietárias.

O objectivo deste estágio deve ser encarado numa perspectiva de prova de conceito. Quer isto dizer que a finalidade do projecto é demonstrar que é possível implementar um sistema que cumpra os requisitos apresentados. Por outro lado, pretende-se também compreender as limitações do sistema no contexto das tecnologias actualmente em uso e perceber de que forma tecnologias futuras poderão ser usadas para as ultrapassar.

O uso de tecnologias, normas e formatos abertos, nomeadamente as normas definidas pelo consórcio OpenGIS, tem sido uma das apostas do grupo de GIS do INESC Porto. Esta aposta tem-lhes permitido desenvolver soluções independentes dos sistemas proprietários em uso nas empresas e instituições. Isto significa que a mesma solução pode ser usada na empresa que usa o sistema proprietário A e na empresa que usa o sistema proprietário B. Por outro lado a aposta nas tecnologias abertas permite também desenvolver sistemas de interface às aplicações proprietárias, possibilitando não só a integração mas também a abstracção dessas aplicações relativamente a outros sistemas.

O projecto m-GIS enquadra-se perfeitamente nesta perspectiva. O uso do formato GML como fonte da informação geográfica usada pelo sistema, e não o formato proprietário A ou B, é exemplo disso mesmo. O uso do GML torna o sistema independente de tecnologias proprietárias e, por isso, mais facilmente integrável com outros sistemas, proprietários ou não.

A componente móvel deste projecto também não é novidade para o grupo de GIS já que um projecto de estágio anterior (*Módulo Remoto para Aquisição de Informação Geográfica num PDA* [Dom02]) desenvolvido neste grupo consistiu na implementação de um módulo remoto capaz de adquirir informação geográfica pontual durante a execução de levantamentos de campo, tendo por base um PDA.

1.3 Organização e Temas Abordados

Para além deste capítulo introdutório este relatório encontra-se estruturado em mais cinco capítulos e três anexos.

O capítulo 2 serve de introdução às tecnologias consideradas relevantes no contexto do problema apresentado. Depois de ler este capítulo, o leitor deve estar apto a perceber o contexto tecnológico do sistema que irá ser desenvolvido e conhecer alguns sistemas e tecnologias que podem vir a ser importantes para o desenvolvimento da nossa solução.

O capítulo 3 descreve a solução proposta para o sistema usando as tecnologias descritas no capítulo anterior. É apresentada a arquitectura geral, as tecnologias que serão utilizadas e a arquitectura do servidor e do cliente. A última secção descreve a integração com um *Web Feature Server*.

O capítulo 4 apresenta os detalhes da implementação do sistema proposto. Será apresentado todo o ambiente de programação, a implementação do servidor m-GIS e do cliente. São ainda descritas algumas optimizações operadas no sistema. Por fim é descrita a implementação de um novo sistema que consiste na integração do anterior com um *Web Feature Server*.

Na primeira secção do capítulo 5 serão apresentados algumas imagens do sistema m-GIS em funcionamento, de forma a termos noção de como o sistema se apresenta ao utilizador. A segunda parte deste capítulo dedica-se à medição das limitações do sistema desenvolvido.

O capítulo 6 apresenta as conclusões que se podem extrair do desenvolvimento e testes efectuados ao sistema m-GIS. Neste capítulo apresentam-se também melhorias ao sistema desenvolvido e outros trabalhos futuros.

O anexo A descreve a tecnologia J2ME, mais concretamente, a tecnologia *Mobile Information Device Profile* (MIDP).

O anexo B apresenta o formato XML para informação geográfica: o *Geography Markup Language* (GML).

No anexo C é descrito o formato *Scalable Vector Graphics* (SVG).

Capítulo 2

Sistemas de Informação Geográfica e a Arena Móvel

O capítulo anterior introduziu o problema que se pretende abordar, os requisitos do sistema a implementar e o seu enquadramento na instituição onde será desenvolvido. Este capítulo serve de introdução às tecnologias consideradas relevantes no contexto do problema apresentado. Depois de ler este capítulo, o leitor deve estar apto a perceber o contexto tecnológico do sistema que irá ser desenvolvido e conhecer alguns sistemas e tecnologias que podem vir a ser importantes para o desenvolvimento da nossa solução.

2.1 Sistemas de Informação Geográfica

Antes de partirmos para uma análise das tecnologias com interesse para o nosso problema, e uma vez que o nosso problema se prende com Sistemas de Informação Geográfica no contexto móvel, convém saber o que é afinal um Sistema de Informação Geográfica (SIG, ou GIS — *Geographic Information System*).

Um Sistema de Informação Geográfica é um sistema de informação que armazena, manipula, analisa e devolve informação geo-referenciada e informação geo-espacial, de forma a auxiliar a tomada de decisão referente ao uso e gestão dos recursos naturais, ambiente, transportes, instalações urbanas, etc.

Os componentes chave de um SIG são os sistemas de computadores, dados geo-espaciais e utilizadores. Um sistema de computadores de SIG consiste em *hardware*, *software* e procedimentos para suportar a captura de dados, processamento, análise, modelação e mostrar dados geo-espaciais.

Os dados geo-espaciais são normalmente classificados em dados gráficos (ou dados geométricos) e atributos (ou dados temáticos). Os dados gráficos têm três elementos: pontos, linhas e áreas, em formato vectorial ou *raster*, que representam uma geometria de topologia, tamanho, forma, posição e orientação.

O papel do utilizador é seleccionar informação pertinente, definir *standards*, desenvolver métodos eficientes de actualização, analisar a saída do SIG com propósitos relevantes e planear a implementação.

2.2 Geography Markup Language (GML)

O GML (*Geography Markup Language*) é uma linguagem XML (*Extensible Markup Language* [W3C03a]), para o transporte e armazenamento de informação geográfica, incluindo tanto as propriedades espaciais como as não espaciais, dos objectos geográficos [OCG01].

O uso desta tecnologia é um dos requisitos do nosso projecto pelo que uma descrição, ainda que breve, se impõe, de forma a introduzir o leitor a esta tecnologia. Uma descrição mais detalhada do GML pode ser encontrada no anexo B.

O GML é uma especificação desenvolvida pelo consórcio internacional OpenGIS. O grande objectivo do GML é fornecer uma plataforma neutra e aberta para a definição de objectos e esquemas geo-espaciais. O GML não é uma linguagem rígida mas antes um sistema que suporta a definição de linguagens de descrição geográfica. Posto muito simplesmente, o GML consiste num esquema (*schema*) XML que pode ser estendido de forma a adaptar-se a situações específicas, mas mantendo sempre uma base comum.

O GML foi desenvolvido com o propósito de atingir vários objectivos. De acordo com a especificação [OCG01]:

- Providenciar uma forma de codificar informação espacial tanto para transporte de dados como para armazenamento;
- Ser suficientemente extensível de forma a suportar uma variedade de tarefas, desde a visualização até à análise;
- Estabelecer a base para o *Internet GIS* de uma forma modular e incremental;
- Permitir a codificação eficiente de geometria geo-espacial;
- Fornecer uma codificação de informação espacial e relações espaciais fáceis de compreender;
- Ser capaz de separar conteúdo espacial e não espacial da apresentação de dados (gráfica, ou não);
- Permitir a fácil integração de dados espaciais e não espaciais;
- Permitir a ligação entre elementos espaciais (geométricos) e outro tipo de elementos espaciais ou não espaciais;
- Fornecer um conjunto de objectos de modelação geográfica de forma a permitir a interoperabilidade de aplicações desenvolvidas independentemente.

O GML está desenhado de forma a suportar a interoperabilidade e fá-lo através da definição de elementos geométricos básicos (todos os sistemas que suportam GML usam os mesmos elementos geométricos), de um modelo de dados comum e de um mecanismo para criação e partilha de esquemas (*schemas*) aplicativos. A maior parte das comunidades de informação facilita a interoperabilidade dos seus sistemas publicando os esquemas GML definidos por si.

O GML foi desenvolvido tendo em conta o princípio da separação entre conteúdo e apresentação. Fornece elementos para a codificação de dados de objectos geográficos sem se preocupar como esses dados serão apresentados. Uma vez que o GML é uma linguagem XML, é facilmente transformável num formato mais adequado à apresentação, seja ela gráfica, sonora, textual, etc.

2.3 Scalable Vector Graphics (SVG)

O SVG é uma linguagem XML para visualização de gráficos bidimensionais. Interessa-nos introduzir aqui esta tecnologia porque este é um dos possíveis formatos de visualização de informação geográfica. Uma descrição mais aprofundada do SVG pode ser encontrada no anexo C.

O SVG (*Scalable Vector Graphics*) é uma linguagem para descrever gráficos bidimensionais em XML. O SVG permite três tipos de objectos gráficos: formas gráficas vectoriais (e.g., polígonos), imagens e texto.

Os desenhos SVG podem ser interactivos e dinâmicos. Através de *scripting*, é possível manipular o DOM (*Document Object Model*) do SVG e ter acesso a todos os elementos, atributos e propriedades. Além disso, a norma define um conjunto de *event handlers* (e.g., `onmouseover` e `onclick`) que podem ser atribuídos a qualquer objecto gráfico do SVG.

A especificação SVG 1.1 (a versão mais recente na altura da escrita deste relatório) está dividida em módulos que fornecem unidades de funcionalidades específicas. Deste modo, os módulos podem ser combinados de forma a constituírem subconjuntos do SVG. A modularização do SVG permite a definição de perfis, compostos por um determinado conjunto de módulos e possivelmente por um conjunto de restrições, ou extensões, aos elementos desses módulos.

Existem, neste momento, três perfis de SVG: o perfil *SVG Full* e os perfis móveis: *SVG Tiny* e *SVG Basic*. O perfil *Full* inclui todos os módulos da especificação SVG. Quando usamos apenas o termo SVG estamos a referir-mo-nos ao perfil *SVG Full*. Os perfis *Tiny* e *Basic* foram definidos tendo como alvo pequenos dispositivos móveis, com limitações de recursos. O perfil *Tiny* é orientado para dispositivos muito limitados enquanto que o *Basic* é orientado para dispositivos de mais alto nível. O perfil *Tiny* é um subconjunto do perfil *Basic*, sendo este um subconjunto do SVG 1.1.

O perfil que nos interessa mais para o nosso projecto é, obviamente, o perfil *SVG Tiny*, uma vez que o nosso sistema é orientado para dispositivos muito limitados.

Sendo tanto o SVG como o GML linguagens XML, o primeiro é facilmente obtido através de aplicação de folhas de estilo XSLT, a partir do segundo.

2.4 O Java Micro Edition (J2ME)

O J2ME (*Java 2 Micro Edition*) é, à semelhança do J2SE (*Java 2 Standard Edition*) ou do J2EE (*Java 2 Enterprise Edition*), uma tecnologia Java da Sun. O J2ME é independente da plataforma (tal como as outras tecnologias Java) e está orientado para dispositivos limitados em termos de recursos e de poder de processamento, quando comparados com os PCs de secretária.

Interessa-nos considerar aqui esta tecnologia porque é o ambiente de programação mais adequado ao nosso problema, uma vez que garante a independência do sistema em relação ao dispositivo.

2.4.1 Visão Geral

O J2ME nasceu da necessidade de adaptar a tecnologia Java aos dispositivos móveis com sérias limitações de recursos quando comparados com os PCs de secretária.

É um ambiente Java direccionado a uma vasta gama de dispositivos móveis, que vai desde *Smart Cards* até *Set-top Boxes*, passando por telemóveis e PDAs (*Personal Digital Assistants*). Consiste num conjunto de especificações organizadas em camadas que permitem abranger um largo leque de dispositivos e tecnologias.

Basicamente, a arquitectura do J2ME está dividida em três camadas como se pode ver na Figura 2.4.1: *máquina virtual, configurações e perfis*.

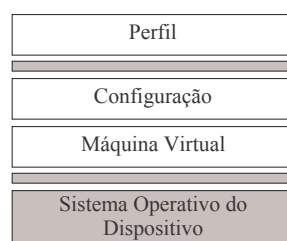


Figura 2.4.1: Arquitectura de um ambiente J2ME

A *máquina virtual* está colocada directamente acima do sistema operativo do dispositivo. É a máquina virtual que define quais são as limitações dos programas que podem correr no dispositivos.

Uma *configuração* é um conjunto de bibliotecas básicas que estão disponíveis para o programador. A configuração define também o nível de funcionalidades e serviços que têm de ser oferecidos pela máquina virtual. Uma configuração é especificada para uma classe horizontal de dispositivos, i.e., uma gama de dispositivos, com diferentes aplicações, mas que partilham algumas características. Por exemplo, a classe dos dispositivos de informação com comunicação *wireless* abrange vários tipos de dispositivos, desde telemóveis, PDAs, *paggers*, etc.

Finalmente, o *perfil* define um conjunto de bibliotecas específicas para uma classe vertical de dispositivos. Seguindo o exemplo anterior, poderíamos ter um perfil para telemóveis, outro para PDAs, etc. As bibliotecas definidas por um perfil são mais específicas do que as definidas pela configuração uma vez que a gama de dispositivos alvo também é menor. Um perfil é sempre especificado para uma determinada configuração, mas uma configuração pode suportar vários perfis.

Neste momento existem apenas duas configurações definidas:

Connected Limited Device Configuration (CLDC) [Sun00a] Esta configuração é usada em dispositivos muito limitados, e.g., telemóveis e PDAs, com capacidade de ligação à rede.

Connected Device Configuration (CDC) Usada em dispositivos com mais capacidade e com melhor conectividade à rede.

A configuração que nos interessa é a CLDC, uma vez que é esta a configuração suportada pelos telemóveis.

O único *perfil* definido até este momento para a CLDC é o *Mobile Information Device Profile (MIDP)* [Sun00b]. O MIDP define alguns requisitos mínimos para o *hardware*, por exemplo,

o ecrã deve ter pelo menos 96x54 pixels, o dispositivo deve ter pelo menos 32Kb de memória volátil para o *heap* do Java *runtime*, etc.

A configuração CLDC em conjunto com o perfil MIDP constituem o ambiente Java que se encontra em grande parte dos telemóveis disponíveis na data de escrita deste relatório.

2.4.2 J2ME Wireless Toolkit (WTK)

O *J2ME Wireless Toolkit* (WTK) é uma ferramenta disponibilizada pela Sun que permite aos programadores testarem as aplicações MIDP. Basicamente, é um conjunto de ferramentas que fornecem um ambiente de emulação para alguns dispositivos, documentação e exemplos de aplicações MIDP.

Esta ferramenta permite configurar o emulador com uma série de parâmetros que afectam o ambiente de execução da aplicação. É possível, por exemplo, configurar qual a versão do protocolo HTTP usado, a velocidade de execução da máquina virtual, a velocidade de transmissão de dados, o tipo de refrescamento do ecrã, etc.

O WTK permite-nos também obter informação sobre os recursos consumidos pela aplicação, em tempo de execução, ou não. É possível saber, por exemplo, em tempo de execução, o estado da memória do dispositivo, que objectos foram alocados, quanta memória livre está disponível, etc.

2.4.3 Algumas Notas Sobre J2ME e Dispositivos Reais

A versão mais recente do MIDP é a versão 2.0, no entanto, até à data do início deste projecto apenas existiam dispositivos que implementavam a versão 1.0. Os primeiros dispositivos MIDP 2.0 foram lançados no Verão de 2003, ou seja, já no final deste projecto.

Alguns dispositivos, nomeadamente telemóveis, limitam o tamanho máximo das aplicações que podem ser instaladas, a cerca de 60Kb. Isto significa que o conjunto dos ficheiros de classes mais os ficheiros auxiliares¹ não pode exceder esse limite, sob pena de o dispositivo não aceitar a aplicação.

A implementação MIDP existente para dispositivos com o sistema operativo Palm (PalmOS) limita a memória das aplicações Java a 64Kb. A Palm anunciou, em Junho de 2003, que os novos dispositivos Palm Tungsten iriam incluir uma máquina virtual licenciada pela IBM. É de esperar, por isso, que as limitações da máquina virtual existente sejam superadas.

Algumas empresas de telemóveis disponibilizam as suas próprias extensões à API MIDP de forma a que seja possível tirar partido de funcionalidades específicas dos seus dispositivos. Essas extensões abrangem áreas como: interface gráfica, acesso aos *phonebooks*, SMS, sons e melodias, etc. O uso destas bibliotecas limita, no entanto, o uso da aplicação aos dispositivos daquele fabricante e colide com o requisito de o sistema ser multi-plataforma.

¹As aplicações MIDP são compactadas em ficheiros JAR, pelo que o limite é normalmente imposto ao tamanho desse ficheiro e não aos ficheiros descompactados.

2.4.4 MIDP e SVG

O projecto TinyLine [Gir03] contém um *toolkit* para trabalhar com SVG Tiny — *TinyLineSVG toolkit*. Esta ferramenta permite incorporar a leitura e *rendering* de imagens em formato SVG em aplicações MIDP 2.0.

Este projecto é, no entanto, muito recente e está ainda pouco “maduro”. Os exemplos fornecidos necessitam de muita memória para executar, mais do que se espera que esteja disponível num telemóvel, por exemplo. Outro ponto desfavorável é o facto de estar implementado para MIDP 2.0, que, na altura da implementação deste projecto, ainda não era suportado por nenhum dispositivo móvel real.

2.4.5 MIDP e XML

Existem várias implementações de interpretadores XML para MIDP. A Tabela 2.4.1 sumariza alguns existentes actualmente.

Tabela 2.4.1 Interpretadores XML para J2ME

Nome	Tamanho	Tipo
kXML	21Kb	<i>pull</i>
ASXMLP	5Kb	<i>push/model</i>
xParse-J	6Kb	<i>model</i>

Todos os interpretadores apontados são potencialmente boas soluções para realizar interpretação de XML em aplicações para MIDP. No entanto o kXML [Enh03b] parece ser o mais “maduro” e estável. Para além disso este interpretador é usado por algumas APIs de SOAP que poderão ser utilizadas para chamar *Web Services* a partir da nossa aplicação.

O xParse é um interpretador do tipo *model*, o que significa que constrói um modelo do documento em memória. Isto pode ser desfavorável em aplicações para dispositivos móveis devido às limitações de memória.

Existem outros interpretadores com implementações muito pequenas a nível de tamanho final das classes, que seriam também bons candidatos para o J2ME. No entanto, uma vez que necessitam de ser portados para J2ME (alguns são fortemente baseados nas APIs do J2SE) não faz sentido, nesta fase, considerá-los.

2.4.6 MIDP e Web Services

O uso de *Web Services* com o J2ME pode passar por uma de várias soluções:

kSOAP O kSOAP [Enh03a] é uma API SOAP para J2ME baseada no kXML. Esta API ocupa cerca de 41Kb já com o kXML incluído.

Wingfoot SOAP O Wingfoot SOAP [Sof03] é outra API baseada no kXML. Este pacote ocupa cerca de 37Kb.

IBM WSTK A IBM disponibiliza um *Web Services Toolkit* [IBM03] baseado no kSOAP e kXML. A versão Java é suportada por dispositivos PocketPC, Palm, e BlackBerry.

Sun JSR 172 Na altura da escrita deste relatório estava em *Public Review* a JSR-172 (*Java Specification Request 172*). Esta especificação pretende trazer dois novos pacotes opcionais para a plataforma J2ME: um para com funcionalidades de interpretação de XML e outro com funcionalidades de *Web Services* baseados em SOAP. No entanto, na altura de implementação deste projecto, esta não era uma opção a considerar uma vez que ainda não estavam definidas as APIs.

O facto de tanto o kSOAP como o kXML serem usados noutros projectos é um indicador da qualidade de ambas as ferramentas. Assim, estas APIs serão a primeira escolha tanto para o uso de *Web Services* como para interpretação de XML.

2.4.7 MIDP e Números de Vírgula Fixa

O formato *SVG Tiny* suporta números de vírgula fixa e uma vez que o CLDC não tem suporte para números de vírgula flutuante é necessário usar classes próprias para trabalhar com estes números. A biblioteca *MathFP* [jT03] define classes para efectuar operações aritméticas com este tipo de números.

2.5 *Web Feature Services e Web Map Services*

Web Feature Services (WFS) e *Web Map Services* (WMS) são duas especificações do consórcio OpenGIS [OCG03].

O WMS especifica o serviço de *Web Map*, ou seja, especifica a interface, *Web*, de um sistema que produz mapas (representações visuais de informação geográfica). A especificação define os tipos de pedidos e respostas que um serviço deste género deve suportar.

O WFS especifica o serviço de *Web Features*, ou seja, define a interface de um sistema que produz informação geográfica no formato GML.

Estas especificações têm por objectivo a interoperabilidade de sistemas de informação geográfica, uma vez que permitem aceder a informação geográfica de uma forma normalizada, residente em qualquer sistema que esteja conforme com a especificação. Daí também o interesse que estas normas têm para o nosso projecto, pelo que vamos fazer uma breve descrição de cada uma delas, e também de um sistema *open-source* que as implementa.

2.5.1 *Web Map Services (WMS)*

Um *Web Map Service* produz mapas de dados geo-referenciados. Um mapa, neste contexto, pode definir-se como uma representação visual de dados geográficos.

A especificação WMS [OCG02b] define três operações: *GetCapabilities*, que retorna meta-informação sobre o serviço, que é uma descrição da informação contida no serviço e parâmetros aceites nos pedidos; *GetMap*, que retorna uma imagem cujo conteúdo e dimensão são definidos pelo cliente; *GetFeatureInfo* (opcional), que retorna informação sobre objectos particulares mostrados no mapa.

A especificação define também a sintaxe dos URLs que invocam cada uma das operações descritas. Está também definida uma codificação XML para a meta-informação do serviço.

Quando é pedido um mapa ao serviço, o cliente pode especificar:

- Que informação deve ser mostrada no mapa (uma ou mais “camadas”).
- O estilo que deve ser aplicado a essas camadas.
- Que porção da Terra deve ser mostrada (*bounding box*).
- Que sistema de projecção, ou sistema espacial de referência, deve ser usado.
- O formato da imagem.
- O tamanho da imagem.
- A transparência e cor de fundo da imagem.

Quando dois ou mais mapas são produzidos com a mesma *bounding box*, sistema espacial de referência e tamanho de imagem, os resultados podem ser combinados de forma a produzir um mapa composto. O uso de fundos transparentes permite que as camadas inferiores permaneçam visíveis. Para além disso, camadas individuais podem ser pedidas a servidores diferentes, ou seja, a especificação WMS permite a criação de redes de Servidores de Mapas distribuídos a partir dos quais os Clientes podem construir mapas personalizados. Um fornecedor particular de um WMS apenas precisa de controlar a sua colecção de dados geográficos. Este esquema contrasta com os sistemas tradicionais que armazenam todos os dados num só lugar e os disponibilizam através da sua interface privada.

2.5.2 *Web Feature Services (WFS)*

Enquanto que um WMS permite a um cliente compor imagens de mapas provenientes de diferentes *Web Map Servers* na Internet, um *Web Feature Service* permite a um cliente obter dados geo-espaciais codificados em GML, provenientes de vários *Web Feature Servers*.

Os requisitos de um *Web Feature Service* são:

- As interfaces devem estar definidas em XML.
- O GML deve ser usado para exprimir objectos geográficos na interface.
- No mínimo, um WFS deve conseguir apresentar objectos geográficos usando GML.
- A linguagem de filtros estará definida em XML.
- O armazém de dados geográficos deve ser opaco para a aplicação cliente. A única vista sobre os dados de um WFS deve ser através das suas interfaces.
- Usar um subconjunto de expressões XPath para referenciar propriedades.

A Figura 2.5.1 (retirada de [OCG02a]) mostra a arquitectura geral de um WFS.

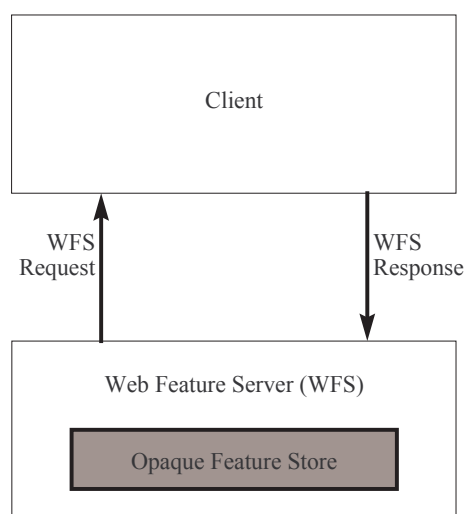


Figura 2.5.1: Arquitectura geral de um WFS

2.5.3 *deegree*

O *deegree* [dee03] é um projecto *open-source* que consiste num conjunto de bibliotecas Java para criação de aplicações geográficas. O desenvolvimento deste projecto tem sido orientado principalmente para a implementação das bibliotecas que permitem desenvolver serviços *Web* (*Web Map Services*, *Web Feature Services*, etc).

O *deegree* é um projecto recente e que continua em desenvolvimento. Estão neste momento implementados um WMS e um WFS, embora ainda com alguns *bugs*.

2.6 Sistemas Existentes

A seguir são apresentados alguns produtos da JShape, semelhantes ao projecto que se pretende implementar. Estes produtos assentam sobre tecnologias proprietárias e não existe muita documentação no que se refere a arquitectura, pelo que apenas são referidos brevemente.

MShape

MShape [JSh03b] é um visualizador de mapas para telemóveis. Funciona com base num servidor de mapas que constrói uma imagem *raster* do mapa e a envia para o dispositivo. Desenvolvido em Java para MIDP.

KShape

KShape [JSh03a] é um visualizador de mapas para Palms. Os mapas são guardados em formato vectorial (um formato próprio da aplicação) e apresentados no Palm. Este sistema foi desenvolvido recorrendo a APIs de teste do CLDC da Sun. Estas APIs foram descontinuadas aquando do lançamento do MIDP.

WShape

WShape [JSh03c] é uma aplicação semelhante ao KShape mas desenvolvida para a máquina virtual Waba (uma máquina virtual de Java não compatível com a da Sun).

Capítulo 3

O Sistema m-GIS

No capítulo anterior foi apresentado o contexto tecnológico em que este projecto se insere. Este capítulo descreve a solução proposta para o sistema, usando as tecnologias descritas anteriormente. É apresentada a arquitectura geral, as tecnologias que serão utilizadas e a arquitectura do servidor e do cliente. A última secção descreve a integração com um *Web Feature Server*.

3.1 Visão Geral

Um sistema deste género levanta imediatamente um problema: que tecnologia usar do lado do cliente? Os telemóveis e PDAs são dispositivos muito heterogéneos em termos de plataformas de suporte a aplicações. Vamos definir como alvo apenas um gama de dispositivos que partilhem uma determinada plataforma? Desenvolvemos a aplicação para as várias plataformas existentes? Ou definimos como alvo, dispositivos com suporte para aplicações Java e desenvolvemos o sistema sobre essa tecnologia?

Outro problema que se nos coloca prende-se com o formato da informação geográfica. O GML é um formato adequado para o armazenamento de informação geográfica, mas não é adequado para visualização. Esta requer uma linguagem que nos permita definir o aspecto, e.g., forma e cor, dos objectos geográficos. O GML não nos permite fazer isso. Isto significa que a informação no formato GML terá de ser transformada num outro formato mais adequado à visualização. Aqui coloca-se o problema de decidir qual o formato a usar. Um formato próprio da aplicação? Um formato gráfico já existente? Do tipo *bitmap* ou vectorial?

Quanto à transformação entre os dois formatos também se coloca uma questão: a transformação deve (ou pode) ser realizada do lado do servidor ou do lado do cliente?

Em relação à comunicação entre cliente e servidor, esta pode ser feita de várias formas. Pode ser feita através de *sockets*, usando um protocolo aplicacional próprio; ou pode ser feita usando um protocolo já existente como o HTTP. A segunda hipótese parece ser a mais indicada devido à maior facilidade de implementação. No entanto, mesmo usando HTTP, podem ser usados diversos mecanismos. Podemos, por exemplo, usar o protocolo SOAP sobre HTTP (*web service*) ou usar HTTP simples usando valores codificados no URL para passagem de parâmetros, por exemplo.

Sejam quais forem as opções que tomemos, podemos desde já esquematizar uma arquitectura de alto nível do sistema a desenvolver, como se mostra na Figura 3.1.1. A informação geográfica

está armazenada no servidor em formato GML. Essa informação será transferida, em GML ou num outro formato, ao cliente. A aplicação do cliente pode decompor-se em dois módulos essenciais: um interpretador e um visualizador. O interpretador será responsável por interpretar a informação recebida do servidor. Muito provavelmente terá por base um interpretador de XML. O visualizador será responsável por desenhar a informação geográfica, e permitir a interacção com o utilizador.

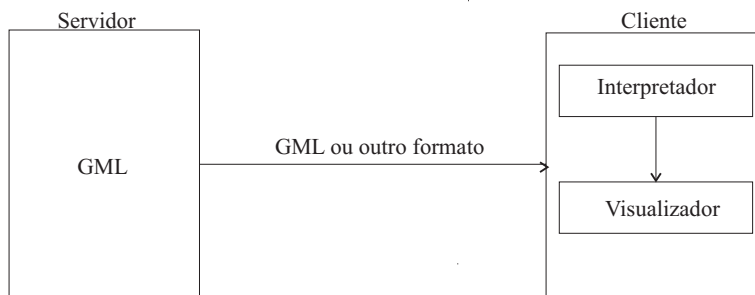


Figura 3.1.1: Arquitectura de alto nível do sistema m-GIS

3.2 Arquitectura Proposta

A solução apresentada tenta conjugar o uso de formatos e tecnologias normalizadas de forma a que o sistema resultante seja tanto quanto possível independente de tecnologias proprietárias, de modo a respeitar os requisitos definidos para o sistema. Da mesma forma pretende-se que o sistema possa ser usado numa vasta gama de dispositivos.

A arquitectura geral do sistema é apresentada em detalhe na Figura 3.2.1.

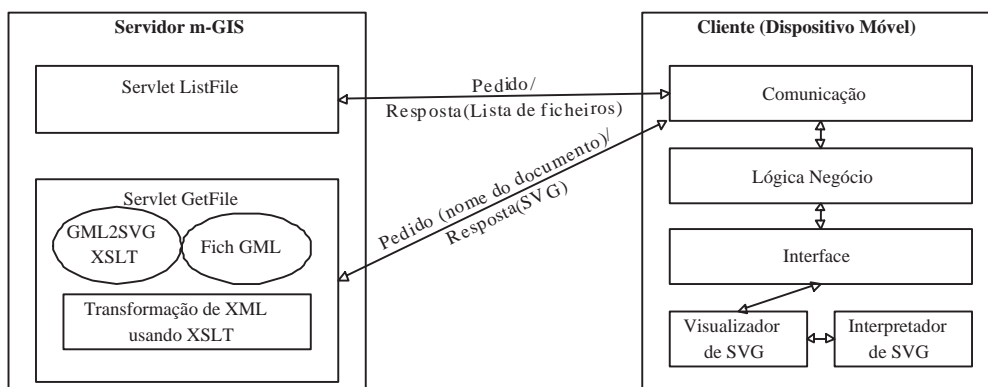


Figura 3.2.1: Arquitectura proposta para o sistema m-GIS

O cliente pode fazer pedidos a duas *servlets*: `ListFile`, para obter a lista de documentos disponíveis no servidor; `GetFile`, para obter o resultado da transformação do documento GML em SVG.

A *servlet* `GetFile` é responsável por transformar um ficheiro XML (neste caso GML) para SVG. O nome do ficheiro a transformar é indicado num parâmetro passado à *servlet*. Para a transformação é necessário um ficheiro XSLT (neste caso um XSLT para transformar GML em SVG) e o próprio ficheiro XML. O resultado da transformação será um ficheiro SVG.

Depois de transformado o GML, o SVG resultante é enviado para o cliente.

Do lado do cliente, os dados SVG são recebidos pelo módulo responsável pela *Comunicação* (o mesmo módulo que iniciou o pedido ao servidor). Os dados são passados por um conjunto de módulos até chegarem ao *Visualizador de SVG*. O *Visualizador de SVG* invoca o *Interpretador de SVG* que irá construir a árvore de elementos SVG. Finalmente o *Visualizador de SVG* irá desenhar o resultado no ecrã.

A aplicação do cliente tem as seguintes funcionalidades:

- Escolher quais os mapas/camadas do mapa que se pretendem visualizar.
- Visualizar o mapa:
 - Deslocar a vista.
 - Aproximar e afastar a vista.
- Pesquisar no mapa.
- Configurar o endereço do servidor que fornece os mapas.

A Figura 3.2.2 mostra os ecrãs disponíveis na aplicação do cliente.

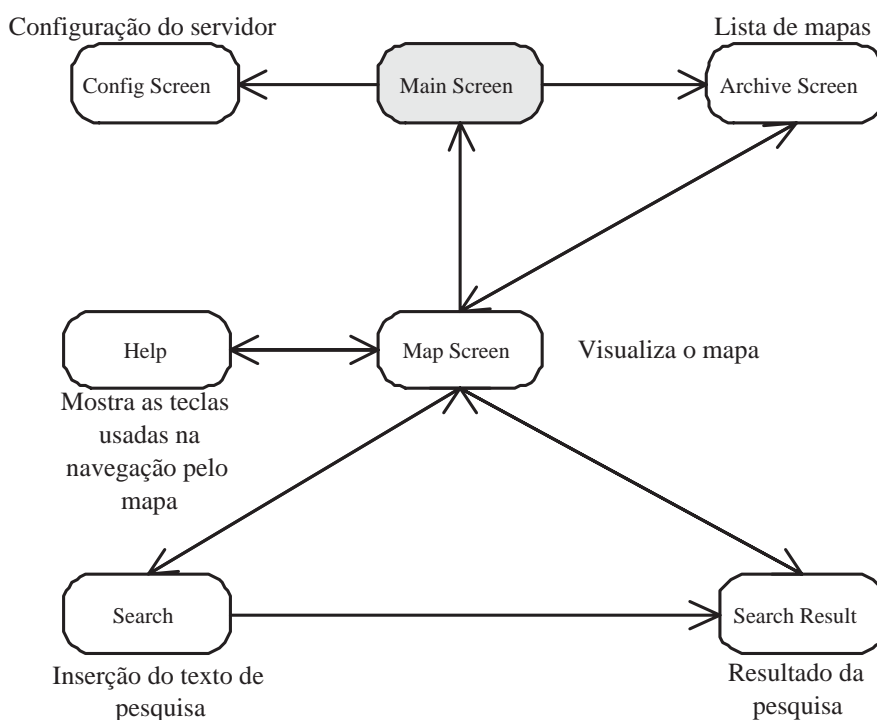


Figura 3.2.2: Diagrama dos ecrãs da aplicação do cliente

O ecrã inicial tem apenas duas opções: *Archive* e *Config*. A opção *Config* permite configurar o endereço do servidor de mapas. A opção *Archive* passa para o ecrã que contém a lista de documentos disponíveis no servidor. O ecrã *Archive* permite escolher quais os mapas, ou camadas de um mesmo mapa, que queremos carregar para o dispositivo. Depois de se escolherem os mapas a visualizar é exibido o ecrã do mapa propriamente dito (durante o carregamento dos

mapas é mostrado em ecrã intermédio de espera). No ecrã do mapa é possível navegar pelo mapa (aproximar, afastar e deslocar a vista), realizar pesquisas sobre o mapa, ver o resultado de uma pesquisa anterior (se tiver sido feita alguma), ver quais as teclas usadas para navegar pelo mapa e voltar aos menus *Main Menu* e *Archive*.

Quando se efectua uma pesquisa é apresentada uma lista de resultados. Neste ecrã é possível escolher qual a entrada que se pretende visualizar. Ao escolher uma entrada, o elemento correspondente, e.g., a rua ou jardim, é centrado no ecrã e destacado a vermelho.

3.3 Tecnologias

O requisito inicial de garantir que o sistema deve ser implementado para dispositivos móveis como telemóveis e PDAs implica quase naturalmente o uso da tecnologia Java, mais concretamente, a tecnologia J2ME.

Como já foi referido, o J2ME é um conjunto de especificações direccionadas a dispositivos limitados quando comparados com os PCs de secretária. O subconjunto desta tecnologia que nos interessa é o perfil MIDP. É esta a especificação seguida pelos vários telemóveis com tecnologia Java, em uso hoje em dia. Assim, a nossa aplicação cliente será desenvolvida usando as bibliotecas MIDP.

Esta escolha não é, contudo, isenta de problemas. Uma vez que o MIDP é dirigido a uma vasta gama de dispositivos, com características e limitações próprias, as bibliotecas disponibilizadas adoptam alguns compromissos no que toca às funcionalidades oferecidas. Uma das limitações mais notada é o facto de as bibliotecas gráficas oferecerem apenas funções básicas.

Consideramos, apesar de tudo, que se justifica o uso desta tecnologia uma vez que é a mais prática no que diz respeito ao desenvolvimento de aplicações para telemóveis e porque é a única que nos permite escrever aplicações independentes da plataforma/dispositivos.

O formato escolhido para visualização no dispositivo é o *SVG Tiny*. É um formato aberto, desenvolvido tendo em atenção as limitações dos dispositivos móveis e é um formato vectorial o que permite realizar operações de *zoom in/out* sem perda de qualidade de imagem. A escolha do SVG como formato de apresentação encaixa também na filosofia deste projecto, que consiste na utilização de formatos e normas abertas. A escolha do SVG deveu-se também ao facto de ser uma linguagem XML, tal como o GML e, por isso, facilmente obtido através de transformação com XSLT. Para além disso, este formato é o formato de visualização “adoptado” pela especificação GML.

Toda a comunicação entre cliente e servidor é feita através de HTTP. O pedidos efectuados ao servidor são pedidos HTTP, com parâmetros passados através de codificação no URL. O uso deste tipo de comunicação é justificado pelo seguinte. Em primeiro lugar, a especificação MIDP 1.0 apenas garante a implementação de comunicação através do protocolo HTTP. A especificação define várias outras formas de comunicação, e.g., *sockets*, datagramas, mas apenas obriga à implementação do protocolo HTTP. Poderíamos ter optado, mesmo assim, por uma forma de comunicação mais avançada, por exemplo, usando SOAP sobre HTTP, vulgo *web services*. No entanto, o uso deste protocolo não nos iria oferecer grandes vantagens uma vez que a interface do servidor é relativamente simples. Além disso, o uso de SOAP iria aumentar a carga de processamento no dispositivo cliente, já severamente limitado a nível de poder de processamento. O uso de SOAP tem ainda outro problema: uma vez que o conteúdo do ficheiro

é encapsulado no envelope SOAP, esse conteúdo só estará disponível quando todo o envelope for recebido. Isto significa que não é possível fazer tratamento do ficheiro à medida que é recebido. O resultado seria maior tempo de espera para o utilizador.

3.4 O Servidor m-GIS

O servidor armazena um conjunto de ficheiros no formato GML. Este conjunto de ficheiros constitui a informação geográfica disponibilizada pelo servidor. Num caso prático, todos os ficheiros conteriam informação sobre uma mesma área geográfica, embora cada um armazenasse apenas um determinado tipo de informação. Cada ficheiro representaria uma camada de informação. Por exemplo, a informação sobre uma cidade pode ser decomposta em eixos de via, edificado, jardins, montes, etc, e cada um destes tipos de informação pode ser armazenado separadamente, em ficheiros. A decomposição pode, obviamente, ser feita de várias formas, e irá permitir ao utilizador escolher apenas a informação que lhe interessa.

O servidor responde a dois tipos de pedidos:

- Pedidos de listagem dos nomes dos ficheiros disponíveis.
- Pedido do conteúdo de um ficheiro (conteúdo transformado em SVG).

O primeiro pedido devolve uma lista de nomes que correspondem aos ficheiros disponibilizados pelo servidor. Deste modo, a aplicação pode apresentar ao utilizador a lista com as camadas de informação disponíveis.

O segundo pedido corresponde ao pedido da informação contida num dos ficheiros da listagem retornada pelo primeiro pedido. Este pedido irá desencadear, no servidor, a transformação da informação em formato GML presente no ficheiro, para o formato SVG. O documento SVG é retornado ao cliente para visualização.

Quando o servidor recebe um pedido de um determinado ficheiro, o conteúdo desse ficheiro é transformado em SVG. O XSLT foi o caminho escolhido para realizar essa transformação. Tanto o GML como o SVG são linguagens XML, sendo, por isso, natural o uso de folhas de transformação XSLT para realizar a transformação de um formato para o outro. Para melhorar a eficiência do servidor na resposta aos pedidos dos clientes, pode ser implementada uma *cache* de XSLT de forma a reduzir o tempo de interpretação do ficheiro XSLT.

3.5 O Cliente

A arquitectura da aplicação cliente¹ está esquematizada na Figura 3.5.1. Esta arquitectura serve vários propósitos:

- Dividir a aplicação em módulos que fornecem funcionalidades específicas.
- Separar a “lógica de negócio” da interface com o utilizador.
- Abstrair a “lógica de negócio” da tecnologia base.

¹Esta arquitectura é baseada num exemplo de [AAA⁺01].

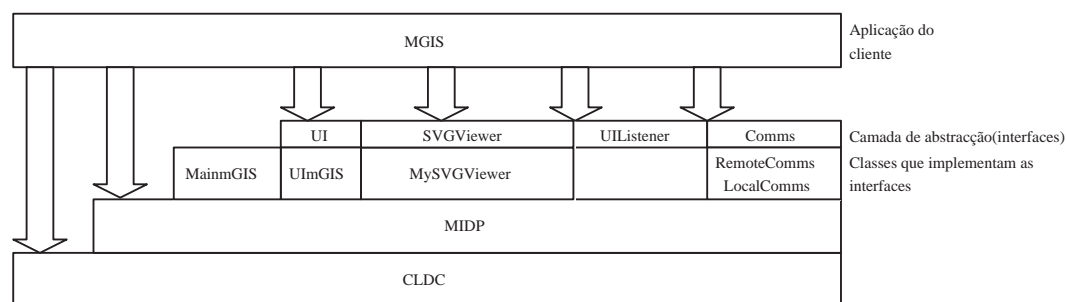


Figura 3.5.1: Arquitectura da aplicação do cliente

O objectivo desta arquitectura é concentrar toda a lógica de negócio na camada identificada por *MGIS*. Esta camada pretende ser independente da tecnologia usada no resto da aplicação, fazendo uso apenas das interfaces definidas na camada de abstracção. Desta forma torna-se possível implementar pequenos detalhes da interface gráfica, por exemplo, recorrendo a bibliotecas específicas de alguns dispositivos, sem alterar a lógica de negócio.

A camada de abstracção é constituída por interfaces e classes abstractas. Isto permite que a aplicação use os serviços disponibilizados por essa camada enquanto que a implementação concreta desses serviços é feita apenas na camada inferior. Desta forma torna-se simples trocar, por exemplo, o visualizador de SVG por outro existente. Da mesma forma é possível implementar o módulo de comunicações através de *Web Services* ou *Servlets* sem alterar o resto do sistema. Esta distribuição em camadas facilita também a possível adaptação do sistema a dispositivos concretos usando as APIs fornecidas pelos fabricantes (e.g. a API da Nokia ou da Siemens). Esta camada divide-se em quatro módulos:

UI O módulo responsável pela interface com o utilizador. Contém funções de alto nível, e.g., `displayWaitScreen()`. A implementação concreta das funções deste módulo fica a cargo do módulo correspondente na camada inferior.

SVGViewer Contém a interface que deve ser disponibilizada pelo visualizador de SVG.

UIListener Representa o *listener* de eventos relacionados com a interface com o utilizador.

Comms A interface de comunicações. Este módulo é usado na comunicação com o servidor.

A camada de classes concretas representa a implementação propriamente dita, da camada de abstracção. É nesta camada que está realmente o código que implementa as funcionalidades definidas pela camada superior.

Uma vez que decidimos usar o SVG como formato de visualização é necessário implementar um interpretador e um visualizador de SVG. A alternativa seria usar um visualizador de SVG já existente, i.e., o *toolkit* TinyLine [Gir03]. No entanto, esta solução tinha vários problemas. O *toolkit* TinyLine está implementado para MIDP 2.0, sendo que até à data não existiam ainda dispositivos deste tipo. Para além disso, é um projecto muito recente e ainda com várias falhas; basta pensar que foi desenvolvido para MIDP 2.0 e que ainda não existiam dispositivos deste tipo, quando foi lançado, pelo que concertiza não foi testado em dispositivos reais. Os testes que efectuámos com este *toolkit* revelaram também que gasta muita memória, mais do que seria de esperar num dispositivo real, actual. Assim sendo, será necessário implementar o nosso próprio visualizador de SVG, o que implica implementar um interpretador de SVG. O interpretador de

SVG terá por base um interpretador de XML já existente: o kXML. O kXML é um interpretador de XML, *open source*, desenvolvido para MIDP 1.0.

3.6 Integração com um WFS

Como já foi referido, depois de serem alcançados os objectivos inicialmente propostos para o projecto deste estágio, enveredámos por uma modificação ao sistema desenvolvido. Essa modificação consistiu na interligação do sistema com um *Web Feature Server*, nomeadamente com o *deegree WFS*. O *deegree WFS* foi escolhido por ser um projecto *open-source*. No lugar do WFS poderíamos ter usado um WMS (o *deegree WMS*), no entanto na altura em que o nosso sistema começou a ser implementado o *deegree WMS* ainda não suportava o formato SVG. Entretanto o *deegree WMS* passou a suportar este formato.

Esta integração vem trazer uma mais valia ao sistema desenvolvido na medida em que permite obter a informação geográfica de uma forma mais flexível através de um serviço normalizado.

O novo sistema é capaz de obter a informação geográfica (GML) através do WFS e apresentá-la ao utilizador, da mesma forma que o sistema antigo, ou seja, transformando a informação para o formato SVG e transmitindo-a à aplicação cliente para visualização.

Esta modificação no sistema obrigou, não só, a uma modificação da parte do servidor do sistema anterior, mas também a uma modificação da aplicação do cliente. O servidor foi alterado por forma a adaptar a fonte da informação geográfica. No sistema anterior a fonte era um conjunto de ficheiros com informação no formato GML, enquanto que no novo sistema a fonte é um WFS. O cliente foi alterado por forma a tirar partido da nova arquitectura do sistema.

A arquitectura geral do novo sistema é apresentada na Figura 3.6.1.

O Servidor

O servidor responde a dois tipos de pedidos:

- Pedidos de listagem das camadas (*layers*) disponíveis no WFS.
- Pedido do conteúdo de uma área. Este pedido deve incluir a delimitação da área pedida.

Tal como no sistema anterior, a informação geográfica pode ser organizada por camadas, de forma a que o cliente apenas visualize o tipo de informação que necessitar. A configuração das camadas disponibilizadas é feita no *deegree WFS*.

Quando o servidor recebe um pedido do conteúdo de uma área, esse pedido é redireccionado ao WFS (com algumas modificações de sintaxe). O WFS devolve a informação geográfica da área pedida, em formato GML, que o servidor m-GIS se encarregará de transformar para SVG e enviar ao cliente.

A articulação do servidor m-GIS com o *deegree WFS* pode ser vista quase como um *Web Map Server*, já que o resultado de um pedido ao servidor m-GIS resulta numa imagem, em formato SVG, representativa da informação geográfica de uma determinada área (obviamente que não pode ser considerado um WMS tal como definido pela especificação *OpenGIS* uma vez que a interface disponibilizada não está de acordo com a especificação).

O Cliente

Também a aplicação do cliente sofreu algumas modificações de modo a tirar partido da nova configuração do sistema.

Na nova aplicação, o utilizador escolhe previamente a área que pretende visualizar sobre um mapa já presente no dispositivo. No protótipo desenvolvido este mapa é composto pelas fronteiras das freguesias do concelho da Maia. O formato escolhido para o mapa inicial foi também o SVG, uma vez que permite adaptar o mapa a qualquer resolução do ecrã do dispositivo e porque nos permitiu reutilizar o visualizador de SVG já desenvolvido. Depois de escolher a área que se pretende visualizar, a aplicação exibe uma lista das camadas de informação presentes no WFS. A lista das camadas disponibilizadas é obtida através de um pedido ao servidor m-GIS, que por sua vez fará o pedido ao WFS. Depois de escolhidas as camadas a visualizar é apresentado o mapa correspondente, sendo que a interacção a partir daqui se faz de forma semelhante ao sistema anterior.

A única diferença na interacção com a nova aplicação reside no facto de ter sido implementada uma nova funcionalidade: a identificação de ruas. Esta nova funcionalidade permite ao utilizador, através da manipulação de um cursor gráfico no ecrã do dispositivo, identificar a rua sobre a qual se posiciona o cursor. Assim, a nova aplicação tem dois modos de funcionamento:

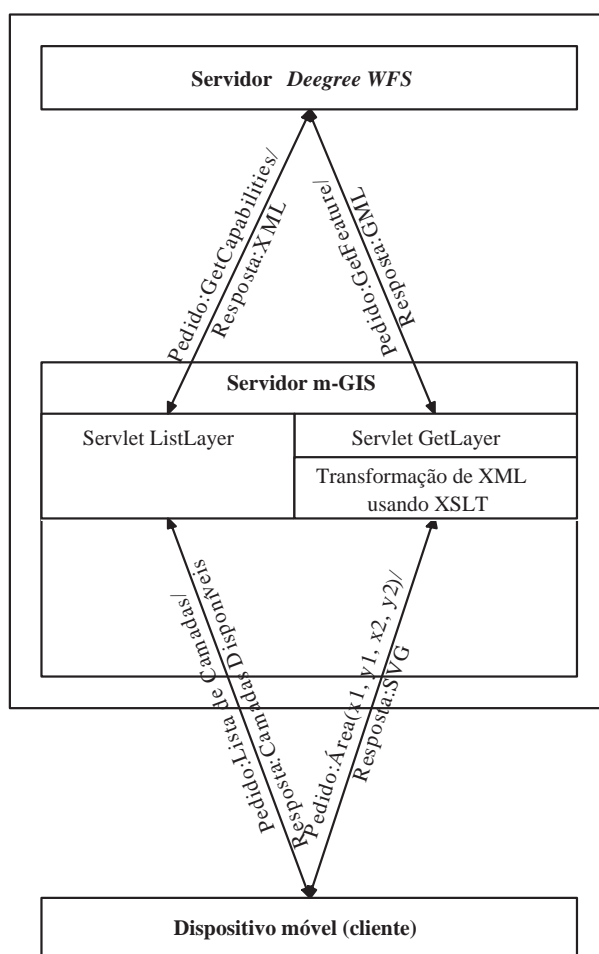


Figura 3.6.1: Arquitectura geral do sistema m-GIS com integração do WFS

Modo mapa O modo de interação já presente na aplicação anterior. Permite a deslocação, aproximação ou afastamento do mapa.

Modo cursor O novo modo de interação. Permite identificar a rua sobre a qual se posiciona o cursor.

Capítulo 4

Implementação do Sistema m-GIS

A solução proposta para o sistema foi apresentada no capítulo anterior em termos de arquitectura geral e de tecnologias utilizadas. Neste capítulo são apresentados detalhes da implementação do sistema proposto. Será apresentado todo o ambiente de programação, a implementação do servidor m-GIS e do cliente. São ainda descritas algumas optimizações operadas no sistema. Por fim é descrita a implementação de um novo sistema que consiste na integração do anterior com um *Web Feature Server*.

4.1 Ambiente de Programação

Esta secção descreve o ambiente de programação usado no desenvolvimento do sistema incluindo as versões de todas as bibliotecas de terceiros usadas.

Java A programação do servidor foi feita usando o J2SDK 1.4.1. O cliente foi programado usando o MIDP 1.0 e o KToolbar 1.0.4_01 para compilação, testes, etc.

Tomcat O *servlet container* usado foi o Tomcat 4.1.

MathFP Foi usada a versão 1.2.2 da biblioteca MathFP para manipulação de números de vírgula fixa.

kXML O interpretador de XML para MIDP usado foi o kXML 1.21.

Saxon A transformação através de XSLT é feita com o processador de XML Saxon versão 6.5.2.

deegree WFS A versão do deegree WFS usado é a versão 1.1.0.

4.2 O Servidor m-GIS

O servidor foi desenvolvido tendo por base o *servlet container Apache Tomcat* e consiste num conjunto de *servlets* e ficheiros auxiliares (ficheiros GML e folha de transformação XSLT).

A arquitectura do lado do servidor é bastante simples. Existem duas *servlets*: `ListFile` e `GetFile`.

A *servlet* `ListFile` retorna a lista dos nomes dos ficheiros GML disponíveis no servidor. São estes ficheiros que o utilizador pode escolher para visualizar no cliente. Basicamente, esta *servlet* lista os ficheiros de um determinado directório configurado no servidor. O directório utilizado para os ficheiros GML é o directório `WEBINF\gml` da *webapp* do Tomcat que foi definida na instalação. No nosso caso a *webapp* chama-se *mgis*. Esta *servlet* aceita um parâmetro: “mode”, que pode ter um de dois valores:

txt O valor por omissão, indica que a lista deve ser um conjunto de cadeias de caracteres (nomes dos ficheiro) separadas por um carácter de mudança de linha;

html Indica que a saída deve ser formatada em html. A lista de ficheiros é apresentada como uma lista de *links* para a *servlet* `GetFile` com um parâmetro igual ao nome do ficheiro. Isto permite-nos usar um *browser* para invocar a *servlet* `GetFile` e obter o resultado da transformação em SVG de um ficheiro. Este modo tem interesse apenas para testar o servidor.

A *servlet* `GetFile` é responsável por transformar o ficheiro GML escolhido para SVG e devolvê-lo ao cliente. A transformação é feita recorrendo à API JAXP da Sun e a uma folha de transformação XSLT. O processador de XML originalmente usado foi o *Xerces*, uma vez que é este o processador usado pelo J2SDK1.4.x, no entanto, devido à existência de alguns *bugs* do *Xerces* encontrados durante o desenvolvimento, optou-se por usar o processador *Saxon* 6.5.2. Esta *servlet* mantém a folha de transformação XSLT em *cache* de forma a diminuir o tempo de interpretação do ficheiro XSLT. Esta *servlet* aceita um parâmetro, “gml”, cujo valor deve ser o nome de um dos ficheiros da lista obtida através da *servlet* `ListFile`.

A Figura 4.2.1 mostra o diagrama de classes do servidor.

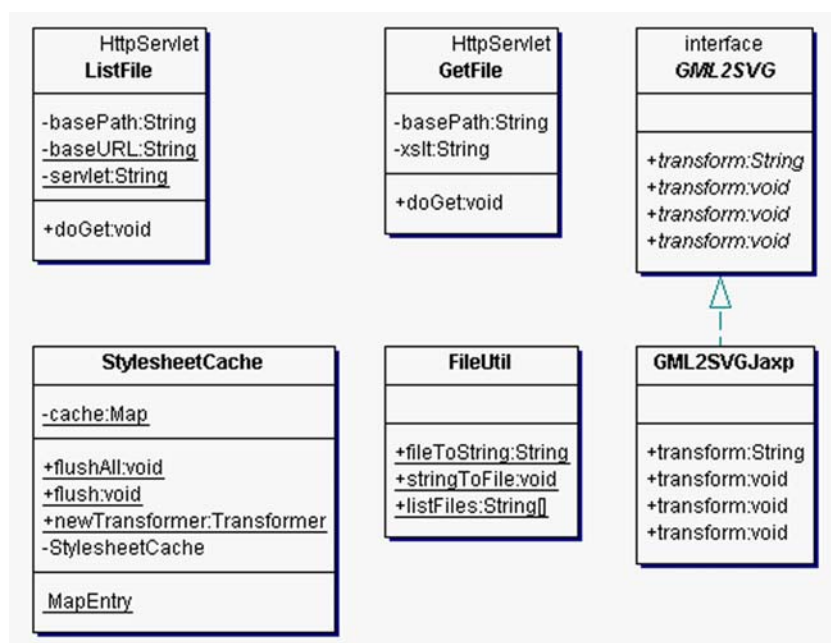


Figura 4.2.1: Diagrama de classes do servidor

Para além das duas classes que representam as *servlets*, `ListFile` e `GetFile`, o diagrama inclui:

GML2SVG A interface de uma classe que fornece funcionalidades de transformação de GML para SVG. Esta interface foi definida porque inicialmente o servidor recorria a um *web service* para efectuar a transformação. Desta forma torna-se fácil alterar a classe que fornece a funcionalidade de transformação.

GML2SVGJaxp A classe que efectua a transformação recorrendo à API JAXP.

FileUtil Uma classe que fornece algumas funções para escrever e ler de ficheiros.

StyleSheetCache Fornece uma *cache* para a folha de transformação XSLT.

Transformação de GML para SVG

A transformação de documentos GML para documentos SVG é feita através de uma folha de transformação (XSLT).

A folha de transformação usada para transformar GML em SVG foi adaptada de um exemplo da *OS MasterMap* [Mas03]. Foram introduzidas algumas modificações básicas, uma vez que a XSLT original se destinava a transformar GML em SVG e não em SVG Tiny. Uma dessas modificações é a conversão dos números para a gama suportada pela norma SVG Tiny. Para além disso foi modificada a forma e o tipo de estilos aplicados aos elementos SVG, mais uma vez devido às restrições impostas pela norma SVG Tiny, tais como a de não suportar estilos CSS.

A norma SVG Tiny suporta números de vírgula fixa na gama -32767.9999 a 32767.9999 . No entanto a informação geográfica, contida nos ficheiros GML, pode usar valores arbitrariamente grandes para representar coordenadas de pontos no espaço. Isto obriga a que a transformação de GML para SVG efectue uma modificação de escala. Na folha de transformação utilizada essa alteração é feita recorrendo aos valores do elemento `<gml:boundedBy>`. Este elemento GML define uma caixa (*bounding box*) dentro da qual todos os pontos definidos no documento GML devem estar contidos. Assim podemos usar o maior valor absoluto das coordenadas dessa caixa, para definir um factor de escala a aplicar a todos os outros valores, de forma a que o resultado esteja dentro da gama pretendida.

Uma vez que queremos distinguir visualmente objectos geográficas diferentes é necessário aplicar diferentes estilos a esses objectos no documento SVG resultante da transformação. Para tal ser possível é necessário que o documento GML contenha informação sobre o tipo de objecto representado, i.e., é necessário que o documento GML indique se determinado objecto é uma estrada, um edifício, um jardim, etc. A norma GML não define elementos para representar objectos como os mencionados atrás, mas define regras para a construção de esquemas GML que definem esses elementos. Assim, foi usado um esquema GML¹ que define, entre outras coisas, elementos para representar edifícios, zonas verdes e vias. A folha de transformação reconhece os elementos definidos no esquema e aplica diferentes estilos conforme o tipo de objecto geográfico representado, e.g., os jardins são pintados a verde.

O Exemplo 4.2.1 mostra um exemplo de um documento GML típico, conforme o esquema desenvolvido. O resultado da transformação desse documento é apresentado no Exemplo 4.2.2.

Podemos ver que um objecto geográfico, por exemplo, uma rua, é representado em SVG usando

¹O esquema GML usado foi adaptado de um já existente criado no âmbito de um outro projecto ([Mon02]) no INESC Porto.

```

<?xml version="1.0" encoding="UTF-8"?>
<Mapa xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:gml="http://www.opengis.net/gml"
  xsi:noNamespaceSchemaLocation="mgismap.xsd">
  <gml:boundedBy>
    <gml:Box>
      <gml:coord>
        <gml:X>-36391.6369611</gml:X><gml:Y>170507.5437507</gml:Y>
      </gml:coord>
      <gml:coord>
        <gml:X>-36012.3401935</gml:X><gml:Y>170770.2912157</gml:Y>
      </gml:coord>
    </gml:Box>
  </gml:boundedBy>
  <MapaMember>
    <Edificado>
      <EdificadoMember>
        <Edificio fid="_1">
          <gml:name>Edificio</gml:name>
          <gml:coverage>
            <gml:Polygon>
              <gml:outerBoundaryIs>
                <gml:LinearRing>
                  <gml:coord>
                    <gml:X>-36263.089</gml:X>
                    <gml:Y>170724.85</gml:Y>
                  </gml:coord>
                  <!-- ... -->
                  <gml:coord>
                    <gml:X>-36263.089</gml:X>
                    <gml:Y>170724.85</gml:Y>
                  </gml:coord>
                </gml:LinearRing>
              </gml:outerBoundaryIs>
            </gml:Polygon>
          </gml:coverage>
        </Edificio>
      </EdificadoMember>
    </Edificado>
  </MapaMember>
  <MapaMember>
    <EixosVia>
      <ViaMember>
        <Eixo fid="_237">
          <gml:name>Avenida do Lidador da Maia</gml:name>
          <gml:edgeOf>
            <gml:LineString>
              <gml:coord>
                <gml:X>-36406.117188</gml:X>
                <gml:Y>170958.203125</gml:Y>
              </gml:coord>
              <!-- ... -->
              <gml:coord>
                <gml:X>-35577.3125</gml:X>
                <gml:Y>170713.4375</gml:Y>
              </gml:coord>
            </gml:LineString>
          </gml:edgeOf>
        </Eixo>
      </ViaMember>
    </EixosVia>
  </MapaMember>
</Mapa>

```

Exemplo 4.2.1: Exemplo de um documento GML

```

<?xml version="1.0" encoding="UTF-8"?><svg id="svgAll" width="100%" height="100%"
  preserveAspectRatio="xMidYMid meet"
  viewBox="-6982.9544 -32767.9999 72.7808 50.4169">
<defs>
  <g id="pointSymbol" overflow="visible">
    <circle cx="0" cy="0" r="1.0" fill="#000000" stroke="#000000"/>
  </g>
</defs>
<g transform="matrix(1 0 0 -1 0 0)" fill="none" stroke-width="0.1" stroke="#000000">
  <g fill="#905050" stroke="#000000" stroke-width="0.1">
    <g id="_1">
      <title>Edifício</title>
      <path d="M-6958.2882,32759.2805z"/>
    </g>
  </g>
  <g fill="none" stroke="#000000" stroke-width="0.1">
    <g id="_237">
      <title>Avenida do Lidador da Maia</title>
      <path d="M-6985.7329,32804.05711159.0339,-46.9665"/>
    </g>
  </g>
</g>
</svg>

```

Exemplo 4.2.2: Resultado da transformação do documento do Exemplo 4.2.1

um elemento `<g>`, sendo que o nome é codificado no elemento `<title>` e a geometria no elemento `<path>`.

O estilo aplicado aos objectos é diferenciado, agrupando objectos do mesmo tipo dentro de um elemento `<g>` e atribuindo a esse elemento o estilo definido para o tipo de objecto. Podemos ver pelo Exemplo 4.2.2 que o edifício está inserido num elemento `<g>` com uma cor castanha (`fill=' '#905050'`) enquanto que a rua não tem preenchimento (`fill=' 'none'`).

O elemento `<g>` de topo serve apenas para realizar uma adaptação do eixo das coordenadas *y*. No SVG, o eixo *y*, tem uma orientação de cima para baixo, enquanto que no GML usado, o eixo *y* tem uma orientação de baixo para cima. Podemos ver que o elemento `<g>` de topo define uma transformação que inverte o eixo das coordenadas *y*.

Praticamente todos os elementos de geometria do GML são transformados usando o elemento `<path>` do SVG. A excepção é o elemento `<Point>` do GML que é convertido para o elemento `<circle>` do SVG. De facto, este elemento, `<circle>`, é apenas definido uma vez no ficheiro SVG e todas as ocorrências de um ponto no documento são transformadas em referências para esse elemento. Isto é feito definindo o elemento `<circle>` dentro do elemento `<defs>` do SVG e atribuindo um “id” ao elemento.

4.3 O Cliente

O desenvolvimento do cliente seguiu a arquitectura definida anteriormente.

O código do cliente distribui-se pelas seguintes *packages* Java:

com.inescporto.mgis Contém as classes principais da aplicação.

com.inescporto.mgis.common Algumas classes partilhadas pelos vários módulos.

com.inescporto.mgis.comms As classes referentes à comunicação cliente/servidor.

com.inescporto.mgis.math Classes relacionadas com transformações geométricas.

com.inescporto.mgis.svg As classes principais do visualizador de SVG.

com.inescporto.mgis.svg.viewer As classes que representam elementos SVG.

com.inescporto.mgis.ui Classes relacionadas com a interface com o utilizador.

com.inescporto.dbg Classes auxiliares para depuração.

Segue-se uma descrição mais pormenorizada das interfaces e das classes envolvidas. A Figura 4.3.1 contém o diagramas de classes correspondente.

Classe MGIS Esta é a classe “mãe” da aplicação. É a classe que agrega as funções da *lógica de negócio*, tendo sido projectada para ser independente da plataforma usada.

Classe MainmGISPhone A classe principal da aplicação. A classe MainmGISPhone estende a classe `Midlet` do MIDP, implementando os métodos de controlo do ciclo de vida da `MIDlet`² (`startApp()`, `pauseApp()`, etc). Para além disto, a principal função é instanciar a classe MGIS.

Interface UI A interface UI contém a definição das funções relacionadas com a interface com o utilizador.

Classe UImGISPhone Esta classe corresponde à implementação da interface UI. É esta classe a responsável por implementar todas as funcionalidades relacionadas com a interface com o utilizador.

Interface UIListener Esta interface define os eventos que podem ser lançados pela classe que define a interface com o utilizador. É a classe MGIS que implementa esta interface de forma a receber os eventos da interface com o utilizador e actuar de acordo.

Interface Comms A interface Comms define os métodos que permitem a comunicação com o servidor.

Classe RemoteComms Esta classe é uma das implementações da interface Comms. Corresponde ao módulo de comunicações da Figura 3.2.1.

Classe LocalComms Esta é uma classe de teste. Neste caso a comunicação não é feita com um servidor, os ficheiros SVG são lidos directamente do dispositivo.

Interface SVGViewer Esta interface define os métodos que o visualizador de SVG deve disponibilizar. É o visualizador de SVG o responsável por interpretar o documento SVG e exibi-lo no ecrã.

Classe MySVGViewer Esta classe corresponde ao visualizador de SVG desenvolvido. A arquitectura do visualizador de SVG é descrita mais à frente.

²*MIDlet* é o nome que se dá a uma aplicação MIDP.

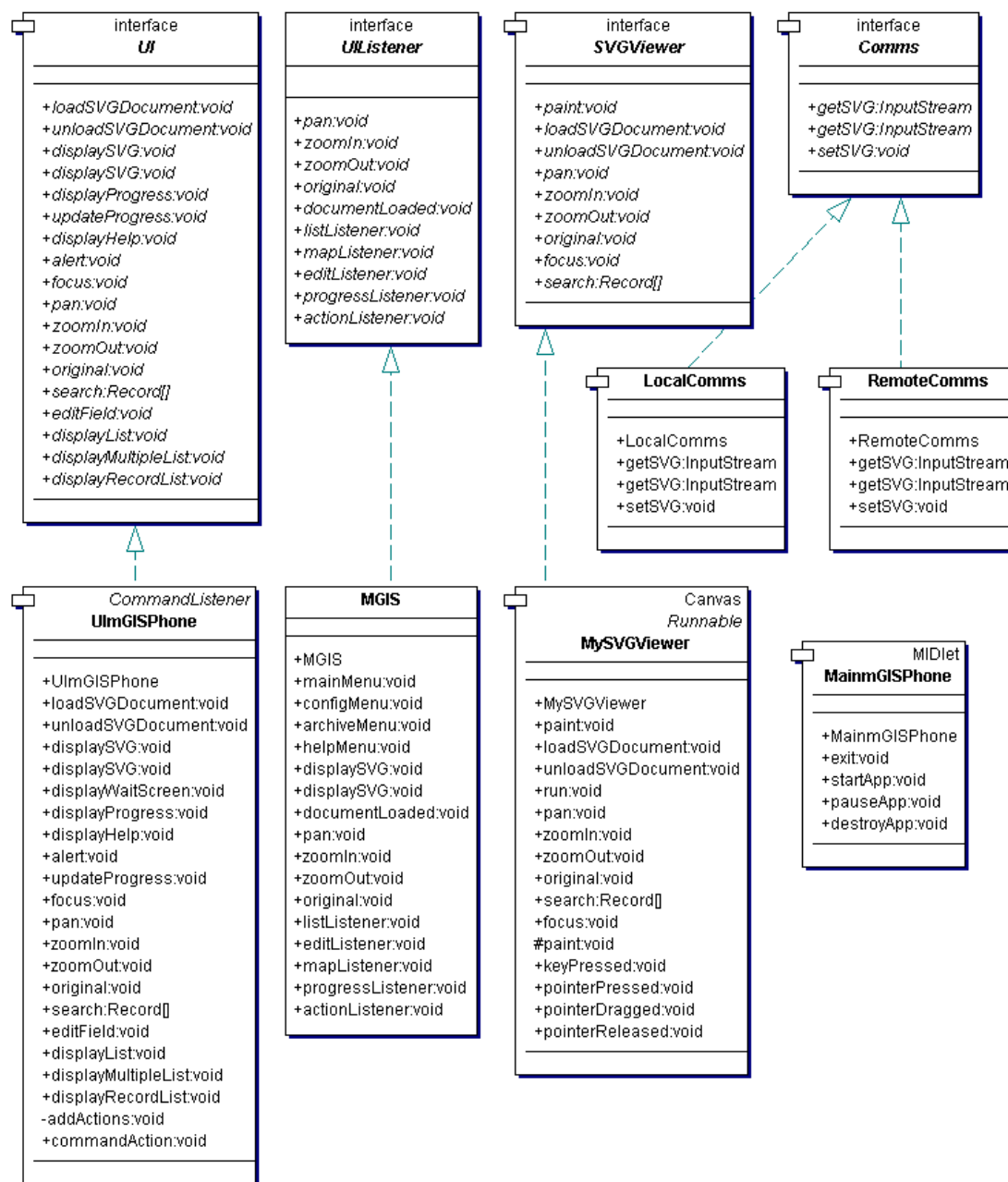


Figura 4.3.1: Diagrama das principais classes da aplicação cliente

4.3.1 Interação

De forma a tornar a aplicação cliente o mais independente possível do dispositivo, não foi feito um mapeamento directo entre os botões do dispositivo e as acções executadas pela aplicação. Poderíamos ter definido, por exemplo, que o botão “1” correspondia à acção “deslocar a vista para a esquerda” e que o botão “2” correspondia à acção “deslocar a vista para a direita”, no ecrã do mapa. No entanto, ainda que este mapeamento fizesse sentido e parecesse “natural” num determinado dispositivo, com uma determinada disposição de teclas, era bem provável que num outro este mapeamento de teclas parecesse pouco natural para o utilizador habituado a utilizar

esse dispositivo.

Exactamente por esta razão, o MIDP define as chamadas *game actions*. As *game actions* não são mais do que “teclas lógicas” que todos os dispositivos MIDP associam às teclas físicas. O MIDP define as seguintes *game actions*: “GAME_A”, “GAME_B”, “GAME_C”, “UP”, “DOWN”, “LEFT”, “RIGHT” e “FIRE”. A atribuição de uma determinada tecla física a uma determinada *game action* fica ao critério do fabricante sendo que a uma determinada tecla pode corresponder no máximo uma *game action*. Desta forma o fabricante pode fazer um mapeamento “natural” entre as teclas físicas e as teclas lógicas. As aplicações podem, em tempo de execução, fazer a correspondência entre as *game actions* e as teclas físicas, tornando aplicação portátil entre dispositivos diferentes.

A aplicação m-GIS define a interface com o utilizador em termos de *game actions*, fazendo, sempre que há um evento do teclado, uma tradução entre o código da tecla física pressionada e a *game action* correspondente.

4.4 O Visualizador de SVG

A Estrutura do SVG

Depois de feita a interpretação do documento, o SVG é representado em memória através de uma árvore de objectos que representam cada elemento do SVG.

As classes usadas para representar o SVG, assim como a classe que efectua a interpretação estão esquematizadas no diagrama de classes da Figura 4.4.1.

Classe SVGNode Todos os nodos da árvore SVG são do tipo SVGNode. É uma classe abstracta que contém a definição de métodos que todos os nodos devem implementar, e.g., *paint()*, para desenhar o nodo no ecrã, *getBB()*, que retorna a *bounding box* do elemento, etc.

Classe SVGCompositeNode É também uma classe abstracta, que representa um nodo que pode conter outros nodos. Todos as classes que representam elementos SVG que podem conter outros elementos devem estender esta classe.

Classe SVGShapeNode Esta classe abstracta representa elementos SVG gráficos, ou seja, elementos como rectângulos, linhas, polígonos, etc. Os objectos do tipo SVGShapeNode são objectos com conteúdo gráfico.

Estas classes são abstractas e por isso não representam elementos SVG concretos. As classes que representam elementos concretos estendem a classe SVGCompositeNode, ou a classe SVGShapeNode. Estas classes são:

SVGSVG Representa o elemento de topo do documento SVG (*<svg>*).

SVGG Representa o elemento *<g>*.

SVGSymbol Representa o elemento *<symbol>*. Este elemento serve para definir elementos que podem ser referenciados no documento SVG. De facto, a especificação SVG Tiny não suporta o elemento *<symbol>*, nem este é usado nos documentos SVG gerados pelo

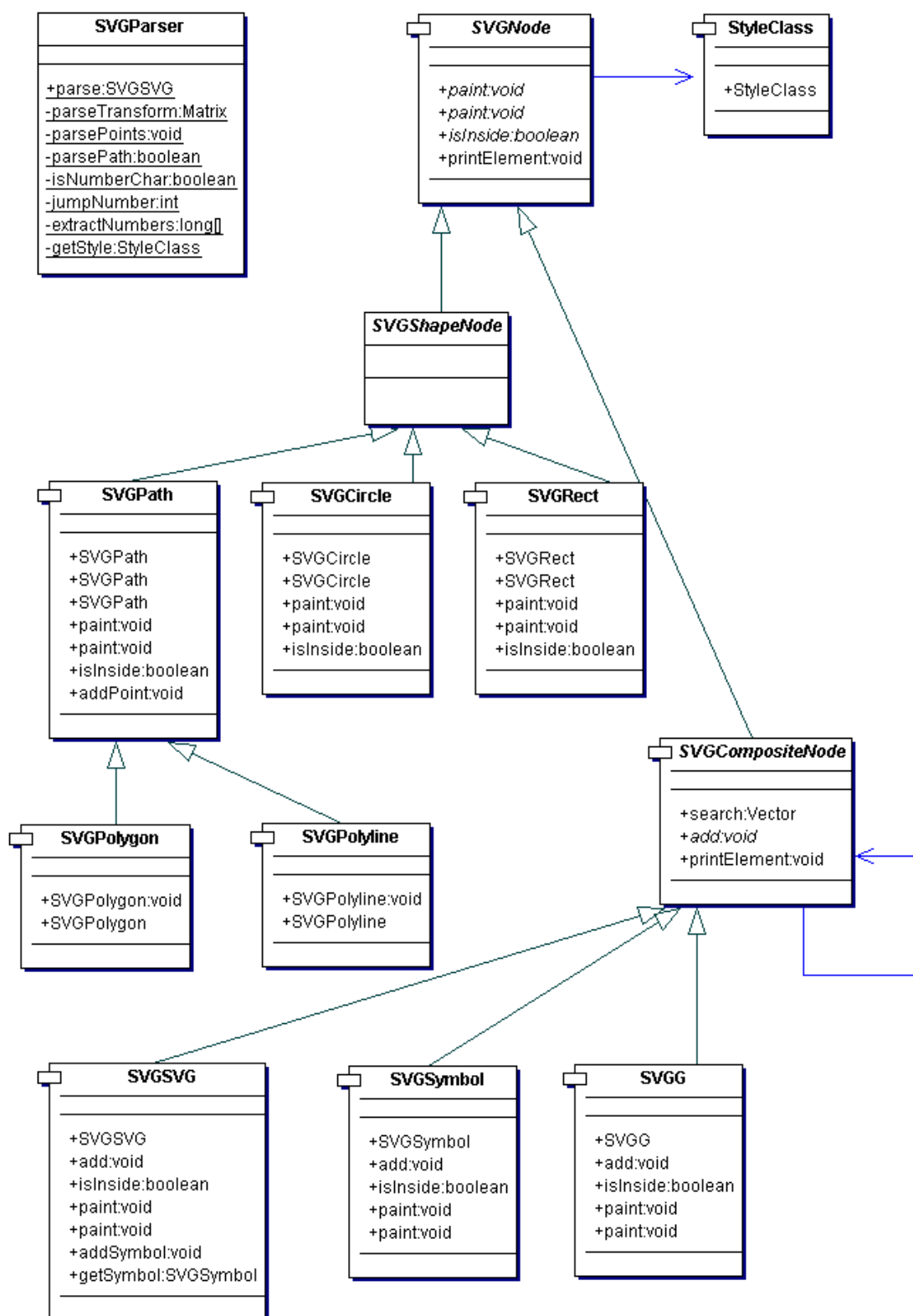


Figura 4.4.1: Diagrama de classes da estrutura do visualizador SVG

servidor. No entanto, por forma a facilitar a implementação, os elementos referenciáveis, contidos no documento SVG gerado são representados, internamente, por esta classe.

SVGPath O elemento `<path>`.

SVGPolygon O elemento `<polygon>`.

SVGPolyline O elemento `<polyline>`.

SVGRect O elemento `<rect>`.

SVGCircle O elemento `<circle>`.

Obviamente que os documentos SVG definem muitos mais elementos, mas apenas foram implementados aqueles necessários para a correcta visualização de um documento SVG gerado pelo servidor m-GIS. Um dos elementos em falta é o elemento `<text>` que permite desenhar texto. Este elemento não foi considerado uma vez que os dispositivos alvo não têm ecrãs com resolução suficiente para mostrarem, por exemplo, os nomes das ruas de um determinado mapa sem o conteúdo do ecrã se tornar ininteligível.

A classe `StyleClass` representa o estilo (cor, preenchimento, etc) que deverá ser aplicado a determinado nodo da árvore. Se não for especificado um estilo para determinado nodo, ser-lhe-á aplicado o mesmo estilo definido para o nodo pai.

O documento SVG do Exemplo 4.4.1 daria origem à árvore de objectos da Figura 4.4.2.

```
<svg viewBox="0.0 0.0 100.0 100.0">
<defs>
  <g id="circulo">
    <circle cx="0" cy="0" r="1.0" />
  </g>
</defs>
<g>
  <rect x="0.0" y="0.0" width="100" height="100"/>
  <path d="M0.0 L 100 100 L 50 50z"/>
</g>
<g>
  <rect x="50.0" y="50.0" width="100.0" height="100.0"/>
</g>
</svg>
```

Exemplo 4.4.1: Um exemplo de SVG

Pesquisa no SVG

A pesquisa na árvore de elementos SVG é feita através do elemento `<title>`. Embora na árvore de objectos não exista nenhum elemento deste tipo, o seu conteúdo é passado, durante a interpretação do documento SVG, para os objectos do tipo `SVGG`. Os objectos do tipo `SVGG` têm um atributo, "title", destinado a guardar o conteúdo do elemento `<title>` associado ao elemento `<g>`.

Quando o utilizador efectua uma pesquisa por um determinado texto, a árvore de objectos é percorrida e todos os objectos do tipo `SVGCompositeNode` são pesquisados pelo atributo

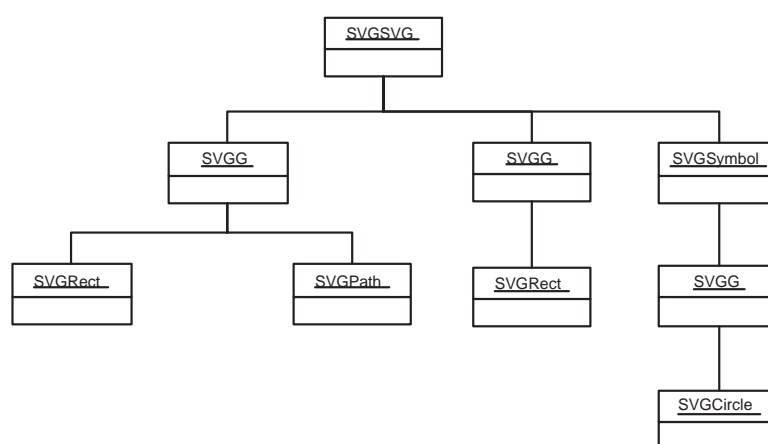


Figura 4.4.2: A árvore de objectos correspondente ao Exemplo 4.4.1

“title”. Os objectos que coincidirem com a pesquisa (basta que o texto a pesquisar apareça na íntegra no atributo “title”) são retornados.

Interpretação do SVG

A classe `SVGParser` é responsável por construir a árvore de objectos correspondente ao documento SVG. O resultado da interpretação será a raiz da árvore (o objecto `SVGSVG`).

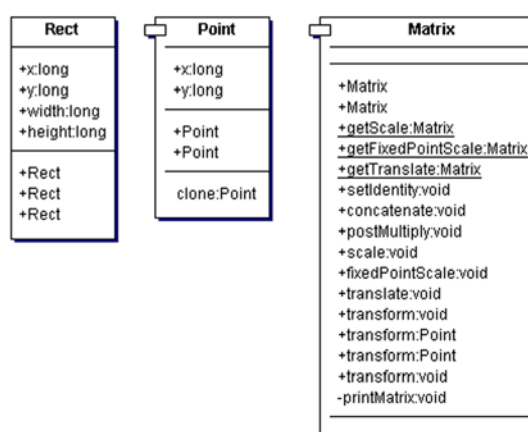
Para realizar a interpretação do documento é utilizado um interpretador do tipo *pull* fornecido pela biblioteca `kXML` [Enh03b]³. Um exemplo do uso do `kXML` para realizar interpretação de XML pode ser encontrado em [Knu02]. A classe `SVGParser` usa o `kXML` para interpretar o documento SVG e construir a árvore de objectos correspondentes aos elementos que a classe reconhece, à medida que a interpretação é feita.

Questões Matemáticas

Uma tipo de operação muito importante para manipular gráficos é a transformação 2D. A forma mais natural de aplicar transformações é através de matrizes de transformação normalizadas (matrizes 3x3, no caso de transformações 2D). Para facilitar a aplicação dessas transformações aos objectos gráficos foi criada a classe `Matrix`. Esta classe contém os métodos tradicionais para manipular matrizes de transformações. A Figura 4.4.3 contém a definição da classe `Matrix` e das outras classes da *package* `com.inescporto.mgis.math`. Grande parte da informação sobre transformações foi retirada do livro *Computer Graphics* [HB97].

Outro ponto importante ao lidar com documentos SVG está relacionado com a precisão dos números usados para representar pontos no espaço. A norma *SVG Tiny* suporta números de vírgula fixa com uma gama de -32767.9999 a $+32767.9999$. Para representar números deste tipo, e uma vez que a API MIDP não suporta números de vírgula flutuante, foi usada a API `MathFP` [jT03] que representa números de vírgula fixa com 64 bits (`long` do Java). De facto, para representar números naquela gama bastavam 32 bits, mas uma vez que o *rendering* dos

³Este interpretador foi ligeiramente alterado de forma a melhorar o desempenho de interpretação de documentos SVG. Estas alterações são descritas na Secção 4.5.1.

Figura 4.4.3: A classe *Matrix*

gráficos implica realizar transformações 2D, é necessário usar mais alguns bits. Todas as coordenadas dos objectos SVG são guardadas no formato MathFP. As transformações 2D também são feitas recorrendo a números neste formato.

4.5 Optimizações

Esta secção descreve algumas optimizações que foram operadas no sistema desenvolvido. Estas optimizações tiveram como objectivo melhorar o desempenho da interpretação dos documentos SVG pelo dispositivo cliente, uma vez que o primeiro protótipo implementado revelou-se demasiado ineficiente.

4.5.1 Interpretação de SVG

Uma vez que o SVG é uma linguagem especificada em XML é natural usar-se um interpretador de XML para realizar a interpretação do SVG. O interpretador escolhido foi o kXML [Enh03b].

O kXML é um interpretador do tipo *pull*. Um interpretador *pull* lê um bocado do documento de cada vez e é a aplicação que controla o interpretador, pedindo o próximo bocado. A vantagem destes interpretadores em relação a interpretadores do tipo *model*, é o menor consumo de memória — os interpretadores do tipo *model* lêem o documento XML de uma só vez e constroem um modelo, em memória, do documento. Em relação aos interpretadores *push*, a diferença reside basicamente no modelo de programação: os interpretadores *push* controlam a interpretação notificando a aplicação, normalmente através de eventos, de que foi encontrada mais uma parte do documento.

Uma vez que o kXML é um interpretador de XML, e não um interpretador de SVG, é necessário construir, em cima do kXML, algo que interprete os elementos e atributos SVG. O Exemplo 4.5.1 mostra parte de um documento SVG constituído por apenas um polígono. Um polígono é especificado através do elemento `<polygon>`, sendo os pontos que constituem o polígono definidos no atributo “points”, como uma sequência de pares de valores numéricos. Neste caso, o kXML é capaz de nos fornecer, por exemplo, o valor do atributo “points”. No entanto para termos acesso aos pontos que constituem o polígono é necessário interpretar esse valor (para o kXML, trata-se apenas de uma cadeia de caracteres).

```

<g id="_N3HE1s3th50">
  <title>Pedroucos</title>
  <polygon points="-7057.0669,30810.5546 -7055.2255,30829.6328
                -7052.8957,30837.5716 -7050.1427,30840.2181
                -7046.4364,30843.9228 -7036.6944,30849.7447
                -7030.9762,30852.0734 -7027.9053,30852.9202
                -7026.4227,30857.2601 -7025.7875,30858.8479
                -7023.6697,30858.5304 -7018.7986,30856.8367
                -7016.9983,30856.8367 -7014.2451,30857.2601
                -7009.6919,30857.8952 -7004.8207,30858.7421
                -6999.9496,30859.483"/>
</g>

```

Exemplo 4.5.1: Parte de um documento SVG típico

Alterações ao kXML

O Exemplo 4.5.1, é um exemplo típico do conteúdo de um documento SVG (documento SVG obtido pela transformação de GML em SVG através da folha de transformação usada no projecto). Podemos ver que grande parte da informação está contida no valor dos atributos dos elementos que definem elementos gráficos.

Devido a esta estrutura típica de um documento SVG, acontece que se torna ineficiente interpretar o SVG. A razão é muito simplesmente a seguinte: o documento SVG tem de ser lido praticamente duas vezes. Primeiro, o kXML tem de fazer a interpretação para devolver à aplicação os elementos, e atributos; depois a aplicação tem de interpretar o conteúdo dos atributos (na maior parte dos casos extrair valores numéricos).

A alteração mais natural consiste em incluir no kXML funcionalidades de interpretação de documentos SVG. Ou seja, obrigar o kXML a interpretar alguns atributos e, em vez de devolver à aplicação uma cadeia de caracteres, devolve, por exemplo, um *array* de pontos. Desta forma a interpretação do ficheiro é feita de uma só vez.

Esta alteração foi levada a cabo modificando-se o código fonte do kXML (o kXML é um projecto *open source*), de forma a interpretar os atributos “points” dos elementos `<polygon>` e `<polyline>` e o atributo “d” do elemento `<path>`. São estes os atributos responsáveis pela maior parte da informação nos nossos documentos SVG.

O diagrama de classes correspondente às classes modificadas no kXML é apresentado na Figura 4.5.1.

A classe mais importante é a classe `XmlParser`. Foram acrescentados três métodos a esta classe:

`parsePoints()` Este método interpreta o atributo “points” dos elementos `<polygon>` e `<polyline>`. Os pontos são guardados num vector (`Vector`) para uso posterior.

`parsePath()` Interpreta o atributo “d” do elemento `<path>`. Tal como no método anterior, os pontos são guardados num vector.

`isNumberChar()` Método auxiliar usado pelos dois anteriores. Indica se determinado carácter pode fazer parte da representação textual de um número.

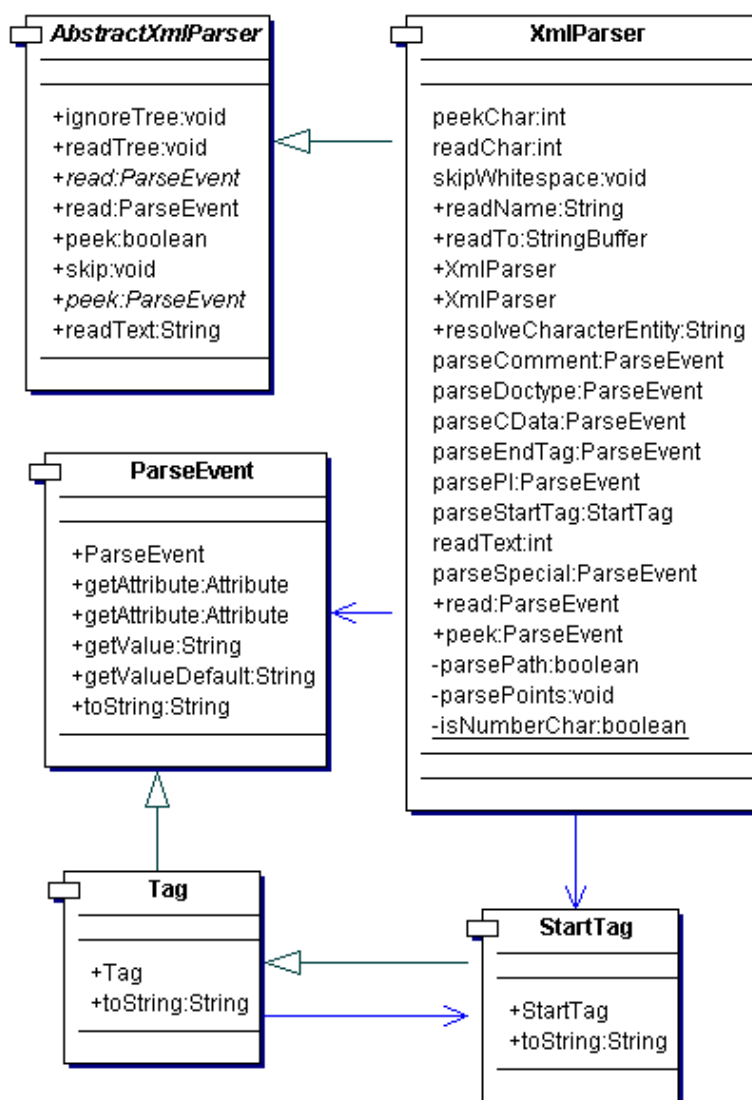


Figura 4.5.1: Diagrama de classes alteradas no kXML

Para além da classe `XmlParser` foi também modificada a classe `ParseEvent` uma vez que é esta a classe que representa um evento de interpretação e que funciona como interface entre o interpretador XML e a aplicação que o usa. A classe foi modificada de forma a devolver o vector de pontos no caso do elemento interpretado ser um dos três referidos anteriormente, `<polygon>`, `<polyline>` ou `<path>`.

Conversão de valores numéricos para o formato MathFP

Uma outra optimização introduzida na interpretação do documento SVG foi a conversão de valores numéricos, em formato de texto, para números de vírgula fixa, formato da biblioteca `MathFP`. Originalmente essa conversão foi deixada sob a responsabilidade da biblioteca `MathFP`. O método `toFP(String s)`, da classe `MathFP`, converte uma cadeia de caracteres para um inteiro de vírgula fixa. O uso deste método, obriga, obviamente, a percorrer a cadeia de caracteres para interpretar os valores e convertê-los para números inteiros.

A optimização introduzida consistiu em delegar a conversão de cadeias de caracteres para inteiros de vírgula fixa, para o kXML. Desta forma a conversão é feita aquando da interpretação inicial do documento e as cadeias de caracteres são lidas apenas uma vez.

Resultados

A Tabela 4.5.1 mostra o tempo de interpretação de alguns ficheiros, antes e depois da introdução das optimizações referidas. Os valores junto ao nome são: o tamanho do ficheiro em kilobytes, o número total de pontos (total de vértices de todos os polígonos) e o número de polígonos definidos no documento.

Tabela 4.5.1 Desempenho da interpretação de documentos SVG.

SVG (Kb/pontos/polígonos)	Tempo de <i>parsing</i>		Ganho
	Antes de otimizar	Depois de otimizar	
maia.svg (138/6385/17)	473 segundos	150 segundos	68%
schools.svg (26/688/133)	65 segundos	33 segundos	49%

4.5.2 Transformação GML para SVG

Codificação de polígonos

Existem várias formas de se definir um polígono em SVG. O formato SVG disponibiliza três elementos que podem ser usados para o efeito: `<polygon>`, `<polyline>` e `<path>`.

O elemento `<polygon>`, tal como o nome indica, é usado para definir uma forma fechada que consiste num conjunto de segmentos de recta ligados entre si. O elemento `<polyline>` é usado, tipicamente, para definir formas abertas que consistem num conjunto de segmentos de recta ligados entre si. Estes dois elementos podem ser definidos a partir de elemento mais geral: `<path>`. O elemento `<path>` tem uma sintaxe muito poderosa que usa o conceito de ponto actual. A partir do ponto actual é possível desenhar-se uma recta ou curva até determinado ponto final, que passa a tornar-se o ponto actual. Também é possível mover-se o ponto actual sem se desenhar nada. Este conceito é usado noutros formatos gráficos como o *Postscript* [Sys99]. Os pontos do elemento `<path>` podem ser especificados em coordenadas absolutas ou relativas e é este facto que nos interessa considerar para otimizar a interpretação do documento SVG.

Uma vez que o elemento `<path>` pode ser usado para se definir formas abertas e fechadas, podemos substituir o uso dos elementos `<polyline>` e `<polygon>` por este elemento, usando, sempre que possível, pontos relativos. Podemos pensar no uso de pontos relativos na definição das formas geométricas como efectuando uma compressão às diferenças dos pontos que definem essas formas. Desta forma o tamanho do documento final será mais reduzido levando a uma interpretação mais rápida. O Exemplo 4.5.2 representa a mesma informação contida no documento do Exemplo 4.5.1, mas usando o elemento `<path>`. É facilmente visível que o tamanho do documento é menor neste segundo caso. Deve ser notado que o espaço em branco entre os pontos e a instrução (a instrução é o “1” entre os pares de coordenadas) pode ser omitida (o espaço em branco foi deixado no exemplo por questões de legibilidade).

Esta modificação na transformação de GML para SVG não é, no entanto isenta de problemas.

```

<g id="_N3HE1s3th50">
  <title>Pedroucos</title>
  <path d="M -7057.0669,30810.5546
    1 1.8414,19.0782 1 2.3298,7.9388
    1 2.753,2.6465 1 3.7063,3.7047
    1 9.742,5.8219 1 5.7182,2.3287
    1 3.0709,0.8468 1 1.4826,4.3399
    1 0.6352,1.5878 1 2.1178,-0.3175
    1 4.8711,-1.6937 1 1.8003,0
    1 2.7532,0.4234 1 4.5532,0.6351
    1 4.8712,0.8469 1 4.8711,0.7409"/>
</g>

```

Exemplo 4.5.2: Parte de um documento SVG típico usando o elemento `<path>`

A principal desvantagem é o tempo extra que irá demorar a transformação, uma vez que será necessário calcular as diferenças entre os pontos e construir o elemento `<path>` com a sua sintaxe mais complexa do que a dos elementos `<polyline>` e `<polygon>`.

Resultados

A Tabela 4.5.2 indica os tamanhos dos ficheiros resultantes da transformação antes e depois de aplicar a optimização à folha de transformação. A compressão referida anteriormente é bem visível nos ficheiros de maior dimensão. O tamanho do documento GML original é indicado apenas para referência.

Tabela 4.5.2 Tamanho do documento resultante das transformações, antes e depois da optimização

Nome	Tamanho (Kb)		
	GML original	SVG (antes de otimizar)	SVG (depois de otimizar)
peq-tudo	296	54	32
peq-edificado	222	40	23
peq-eixos	26	6	5
peq-jardins	40	8	5
maia	498	138	95

A Tabela 4.5.3 indica o tempo que demora a fazer a transformação de GML para SVG nos casos antes e depois da optimização da transformação. Como seria de esperar a transformação demora ligeiramente mais tempo no caso da transformação optimizada (a palavra optimizada neste contexto refere-se à optimização do documento SVG resultante, em termos de espaço ocupado, obviamente).

Tabela 4.5.3 Tempo de transformação dos ficheiros antes e depois da optimização

Nome	Tempo (segundos)	
	Antes de otimizar	Depois de otimizar
peq-tudo	5,6	6,6
peq-edificado	5,0	5,6
peq-eixos	2,3	2,6
peq-jardins	2,6	2,9
maia	5,8	16,1

Esta optimização realizada à transformação GML para SVG apenas nos interessa se a diminuição do tamanho final do documento SVG compensar em relação ao acréscimo do tempo de transformação. Por outras palavras o que nos interessa é que o tempo de espera do utilizador, desde que faz o pedido ao servidor até visualizar o mapa, diminua. A Tabela 4.5.4 indica-nos exactamente os tempos de interpretação do documento, incluindo o tempo de transformação. São estes valores que interessa reduzir. Os dados da tabela revelam que o tempo de espera do utilizador diminui efectivamente.

Tabela 4.5.4 Tempo de interpretação dos ficheiros antes e depois da optimização.

Nome	Tempo (segundos)	
	Antes de optimizar	Depois de optimizar
peq-tudo	86	58
peq-edificado	65	42
peq-eixos	9	8
peq-jardins	13	9
maia	173	155

4.6 Integração com um WFS

Como já foi abordado, a segunda fase deste projecto consistiu na integração do sistema desenvolvido na primeira fase, com um *Web Feature Server*.

A seguir são descritas as principais alterações efectuadas ao sistema anterior.

4.6.1 O Servidor

A Figura 4.6.1 apresenta o diagrama de classes correspondente ao servidor.

O servidor é composto por duas *servlets*: `ListLayer` e `GetLayer`. Estas duas *servlets* são análogas às *servlets* `ListFile` e `GetFile`, respectivamente, do servidor antigo.

A *servlet* `ListLayer` retorna ao cliente a lista de camadas (*layers*) de informação disponíveis no WFS. Esta lista é pedida directamente ao WFS, fazendo um pedido de *GetCapabilities*, que retorna, entre outras informações, as camadas configuradas no WFS. Este pedido é representado pela classe `CapabilitiesRequest`. O resultado deste pedido (todos os pedidos e respostas ao WFS são feitos em XML) é interpretado e encapsulado pela classe `Capabilities`.

A *servlet* `GetLayer` é responsável por obter do WFS o GML representativo de uma determinada área (a área é definida em parâmetros da *servlet*), transformá-lo em SVG e enviar a resposta ao cliente. Os parâmetros aceites por esta *servlet* são os seguintes:

xmin A menor coordenada x do rectângulo que representa a área que se pretende visualizar.

xmax A maior coordenada x do rectângulo que representa a área que se pretende visualizar.

ymin A menor coordenada y do rectângulo que representa a área que se pretende visualizar.

ymax A maior coordenada y do rectângulo que representa a área que se pretende visualizar.

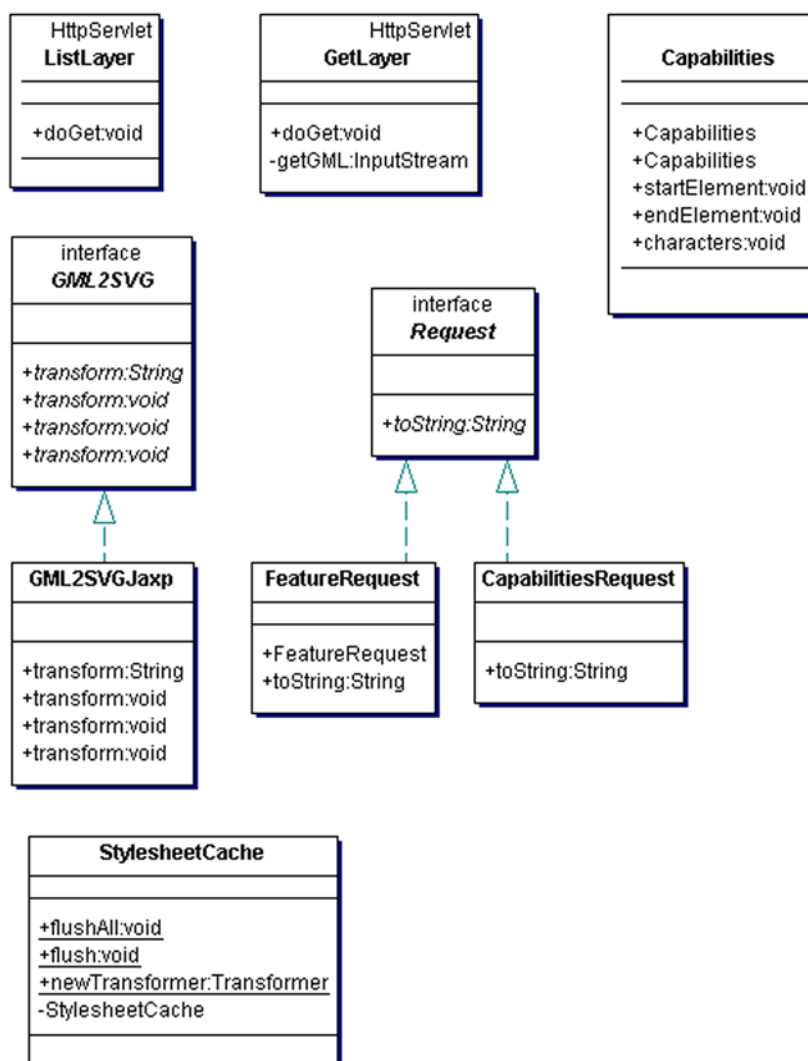


Figura 4.6.1: Diagrama de classes do servidor m-GIS na nova configuração

layers Uma lista de camadas que se pretende visualizar. Os nomes das camadas são separados por vírgulas.

Estes parâmetros são encapsulados num pedido do tipo *GetFeature* que é feito ao WFS. O resultado deste pedido, representado pela classe *FeatureRequest*, ao WFS é um documento GML que será depois, à semelhança do que se passava no servidor antigo, transformado em SVG e devolvido ao cliente.

Tal como no servidor antigo, a folha de transformação XSLT é mantida em memória através de uma cache, implementada pela classe *StylesheetCache*.

Transformação de GML para SVG

A transformação de GML para SVG no caso do novo servidor sofreu algumas alterações:

Esquema GML Uma vez que o esquema GML usado é diferente do anterior, já que neste

caso o GML é gerado por um WFS, houve necessidade de alterar o XSLT de forma a reconhecer os novos elementos representativos das camadas de informação, por exemplo.

Bounding box No XSLT do servidor antigo, a *bounding box* do SVG correspondia directamente à *bounding box* do GML (embora com uma adaptação de escala). Neste caso não podemos proceder do mesmo modo. Um pedido ao WFS resulta num documento GML que contém todos os objectos geográficos que intersectam a área pedida. Isto significa que a *bounding box* do documento GML não corresponde exactamente à área pedida, mas é normalmente superior. Esta situação é ilustrada na Figura 4.6.2, a área pedida corresponde ao rectângulo, a área recebida é toda a figura. Assim sendo é necessário transmitir à folha de transformação a *bounding box* original (a área realmente pedida) de forma a que a *viewport* do SVG corresponda à área que o utilizador pretende visualizar.

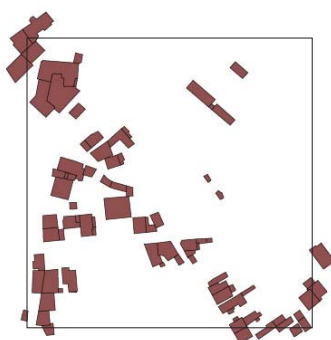


Figura 4.6.2: Diferença entre a área pedida e a área recebida usando um WFS

Configuração do *deegree WFS*

A configuração do *deegree WFS* consistiu, basicamente, na definição das fontes de dados a serem usadas pelo servidor. No nosso caso o servidor foi configurado de forma a obter a informação geográfica a partir de ficheiros do tipo *ESRI shapefiles*. Este é um formato da ESRI [ESR03], uma empresa de software de SIG, usado em vários dos seus produtos, que o *deegree* suporta como fonte de dados.

Foram configuradas quatro camadas de dados, que correspondem a quatro ficheiros *shapefile*, com informação do concelho da Maia: uma camada com os limites das freguesias, uma camada com os eixos de via, uma com o edificado e uma com as farmácias.

4.6.2 O cliente

As principais alterações à aplicação cliente deram-se ao nível das funcionalidades disponibilizadas. A arquitectura geral da aplicação manteve-se igual à do sistema anterior.

As novas funcionalidades da aplicação são:

Escolha da área a visualizar A nova aplicação permite ao utilizador escolher a área que pretende visualizar dentro de um mapa pré-carregado no dispositivo.

Identificação de ruas A nova aplicação permite ao utilizador identificar uma determinada rua, posicionando um cursor gráfico sobre a rua.

A implementação da primeira funcionalidade é concretizada numa nova classe: `AreaSelector`. O que esta classe faz é simplesmente mostrar uma imagem e permitir ao utilizador manipular um rectângulo sobre essa imagem de forma a poder escolher uma determinada área.

A identificação de ruas no mapa, é feita percorrendo todos os objectos do tipo `SVGPath` e calculando a distância de todos os segmentos de recta desse objecto, ao ponto em que se encontra o cursor gráfico. É retornado o objecto `SVGG` que contém o `SVGPath` mais próximo do cursor.

Capítulo 5

Exploração e Testes

O capítulo anterior descreveu o processo de desenvolvimento do nosso sistema. Foram apresentados detalhes de implementação tanto do primeiro sistema desenvolvido como do segundo (com integração de um *Web Feature Server*). Na primeira secção deste capítulo serão apresentados algumas imagens do sistema m-GIS em funcionamento, de forma a termos noção de como o sistema se apresenta ao utilizador. O sistema ilustrado corresponde ao sistema com integração de um *Web Feature Server*. A segunda parte deste capítulo dedica-se à medição das limitações do sistema desenvolvido.

5.1 O Sistema m-GIS em Acção

Esta secção apresenta o sistema desenvolvido em funcionamento a partir de um emulador de telemóvel do *Wireless Toolkit* da Sun. O sistema apresentado corresponde ao sistema com integração do WFS uma vez que a interface com o utilizador deste é mais complexa do que a do sistema anterior.

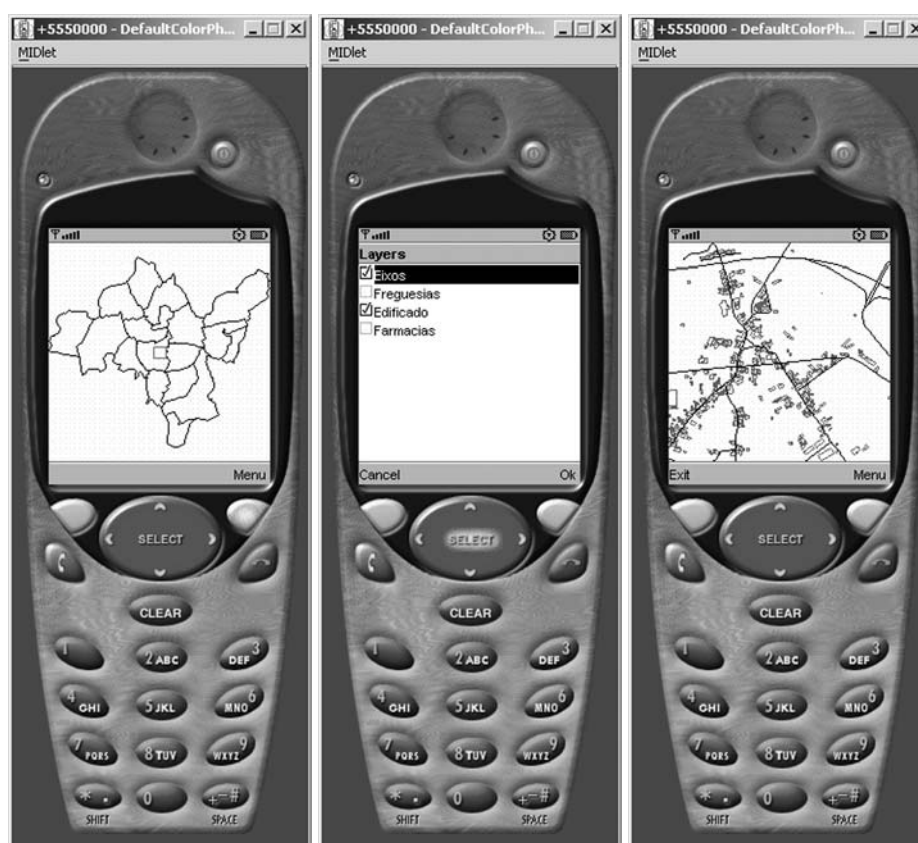
A Figura 5.1 ilustra uma sequência de interacção do utilizador com o sistema m-GIS a partir de um telemóvel. A interacção resume-se a escolher a área a ser visualizada, as camadas de informação e, por fim, a navegação pelo mapa.

A Figura 5.1(a) mostra o ecrã inicial apresentado ao utilizador. Podemos ver o mapa das freguesias do concelho da Maia — este mapa encontra-se pré-carregado no dispositivo, ou seja, faz parte da aplicação instalada. Podemos ver também um pequeno rectângulo que pode ser manipulado, i.e., aumentado, diminuído ou deslocado, através dos botões “GAME_A”, “GAME_B” e botões de direcção, respectivamente; este rectângulo corresponde à área seleccionada. Depois de seleccionada a área (pressionando o botão “Select”) é apresentado o ecrã das camadas.

A Figura 5.1(b) ilustra o ecrã em que o utilizador pode escolher as camadas de informação que pretende visualizar no mapa. A lista é composta por itens que podem ser marcados ou desmarcados. Depois de escolhidas as camadas é apresentado o mapa com a informação pretendida¹.

O terceiro ecrã (Figura 5.1(c)) corresponde ao mapa propriamente dito. Neste ecrã o utilizador pode navegar pelo mapa usando os botões “GAME_A” para aproximar a vista, “GAME_B” para afastar, os botões de direcção para deslocar a vista e o botão “FIRE” para colocar o mapa na

¹De facto existe ainda um outro ecrã que é exibido entre estes dois. Este ecrã mostra o progresso no carregamento do mapa e é, no fundo, um ecrã de espera.



(a) Ecrã inicial

(b) Camadas disponíveis

(c) O mapa

Figura 5.1: Sequência de interacção com o sistema m-GIS

posição original. O “modo cursor”, que o utilizador pode utilizar para identificar ruas, é acedido através do botão “GAME.C”, que alterna entre os dois modos - “modo cursor” e “modo mapa”. Neste modo, ilustrado na Figura 5.2(b), o utilizador pode deslocar um cursor gráfico, utilizando os botões de direcção, e identificar uma rua que esteja sob o cursor (ou suficientemente perto), utilizando o botão “FIRE”. A partir do ecrã do mapa é possível aceder a várias opções, como se pode ver na Figura 5.2(a):

Search Para efectuar uma pesquisa sobre o mapa. A Figura 5.2(c) ilustra o resultado de uma pesquisa por “rua”.

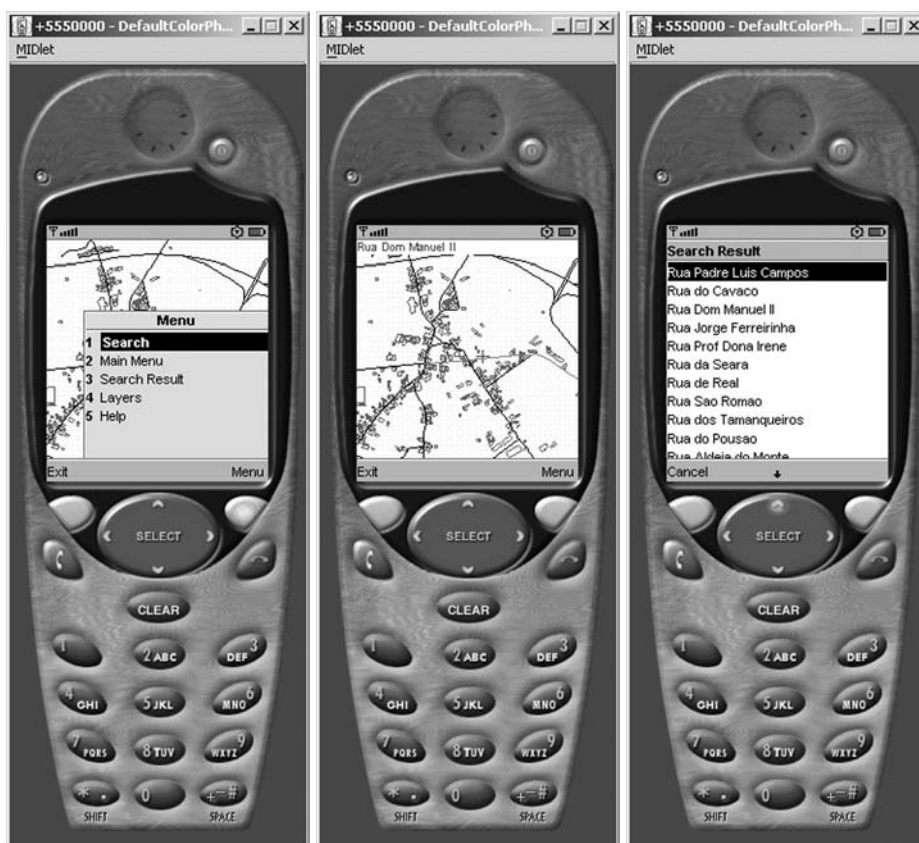
Main Menu Para voltar ao ecrã inicial.

Search Result Para voltar a mostrar o resultado da última pesquisa.

Layers Para voltar ao ecrã de selecção de camadas.

Help Para mostrar o ecrã de ajuda. Este ecrã mostra o mapeamento entre as teclas e as acções correspondentes.

A aplicação do cliente não é uma aplicação apenas para telemóveis. Qualquer dispositivo que siga a especificação MIDP pode executar a aplicação. Na Figura 5.3 podemos ver duas imagens da nossa aplicação a funcionar num PDA HP Jornada. As imagens foram obtidas usando a



(a) Opções do mapa

(b) “Modo cursor”

(c) Resultado de uma pesquisa por “rua”

Figura 5.2: Algumas imagens do sistema m-GIS

aplicação *Remote Display Control*, que permite visualizar o ecrã e controlar o PDA através de um PC de secretária.

5.2 Limites do Sistema m-GIS

Esta secção apresenta os resultados de algumas medições efectuadas sobre o sistema m-GIS. O sistema é analisado segundo vários eixos: memória consumida, tamanho máximo do mapa visualizado e tempos de transformação e de interpretação dos documentos SVG.

Os testes foram efectuados usando um computador com processador AMD Athlon a 1 GHz e com 256 Mb de memória RAM. Neste computador foram instalados o servidor WFS e o servidor m-GIS. O emulador de telemóvel usado foi também executado nesta máquina. No caso do PDA, foi usado um HP Jornada 548 Pocket PC com 32Mb de memória e com a máquina virtual Personal Java [Mic03] da Sun instalada. Neste caso não existe emulação. A MIDlet foi executada recorrendo à ferramenta ME4SE [ME403] que permite executar MIDlets em ambientes Java “normais”. A ligação do PDA com o computador onde se encontrava o servidor foi feita através de USB.

Para os testes efectuados com o *Wireless Toolkit* foi usada a seguinte configuração:

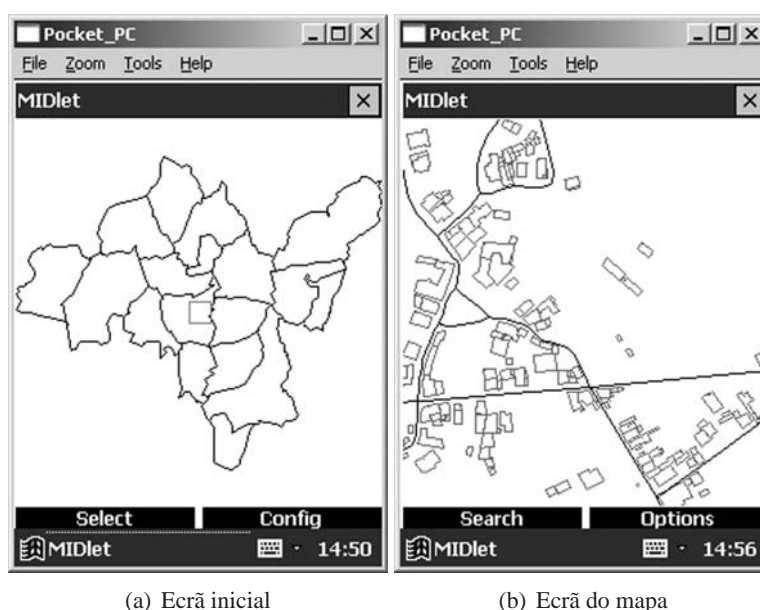


Figura 5.3: Algumas imagens do sistema m-GIS num PDA

Network Configuration

Http Version - HTTP/1.1

Performance

Graphics primitives latency - 0 milliseconds

Display Refresh - Immediate

Enable VM speed emulation - 100 bytecodes/millisecond

Uma das características dos dispositivos MIDP é a quantidade de memória reservada para as aplicações Java. A quantidade de memória do dispositivo impõe um limite à quantidade de informação geográfica que pode ser visualizada pelo dispositivo, ou seja, limita o tamanho do mapa visualizado.

Tabela 5.2.1 Dimensões máximas dos mapas de acordo com a memória disponível

Memória (Kb)	Dimensão do mapa (m)	Camadas	Tamanho do documento SVG (Kb)
192	1033x1029	eixos	29.1
256	1549x1544	eixos	55.0
320	2324x2316	eixos	95.5
384	2840x2831	eixos	138.2
448	3098x3088	eixos	159.7
512	3615x3603	eixos	205.2
256	258x257	edificado	22.7
320	516x515	edificado	65.0
448	775x772	edificado	103.4
256	258x257	eixos + edificado	34.7
384	516x515	eixos + edificado	78.3
448	775x772	eixos + edificado	124.3

A Tabela 5.2.1 apresenta, para uma determinada quantidade de memória, o tamanho máximo

que o dispositivo consegue visualizar. Estes valores foram obtidos com o emulador de telemóvel do WTK. Apenas foram testados mapas com três tipos de informação: eixos de via, edificado e eixos de via mais edificado. Podemos ver que com 192Kb conseguimos visualizar mapas dos eixos de via de dimensão até cerca de 1x1 quilómetros. Para visualizar mapas de edificado precisamos de, pelo menos, 256Kb de memória.

Um dos factores mais limitativos do sistema m-GIS é o tempo de transformação e de interpretação dos documentos SVG. A Tabela 5.2.2 apresenta alguns tempos medidos com o emulador de telemóvel do WTK. A tabela apresenta a dimensão do mapa, as camadas presentes no mapa, o tempo que demora a transformação de GML para SVG, o tempo que o dispositivo demora a efectuar a interpretação do documento², o tempo que o mapa demora a ser desenhado no ecrã do dispositivo e o tamanho do documento SVG.

Tabela 5.2.2 Tempos de transformação e de interpretação

Dimensão mapa (m)	Camadas	Tempos (segundos)			Tam. SVG (Kb)
		XSLT	Interpretação	Rendering	
258x257	eixos	0.8	20.5	3.2	13.0
516x515	eixos	0.8	22.6	3.6	14.2
775x772	eixos	1.0	33.8	5.4	21.9
1033x1029	eixos	1.1	46.3	7.3	29.9
1291x1287	eixos	0.5	57.3	8.9	37.5
1549x1544	eixos	1.7	83.1	13.0	55.4
1807x1801	eixos	2.5	108.9	16.9	72.4
2066x2059	eixos	25.5	128.2	19.5	85.3
2324x2316	eixos	15.2	147.0	22.7	97.8
2582x2574	eixos	49.1	175.2	25.9	117.1
2840x2831	eixos	72.7	211.5	30.4	141.5
3098x3088	eixos	98.4	244.8	34.1	163.5
3357x3346	eixos	147.0	285.5	38.4	190.2
3615x3603	eixos	160.0	313.7	41.3	210.1
258x257	edificado	0.3	40.6	6.5	23.3
516x515	edificado	0.9	111.6	14.6	66.5
775x772	edificado	27.4	175.1	17.2	105.9
258x257	eixos+edificado	2.3	59.3	9.7	35.6
516x515	eixos+edificado	2.7	130.0	18.1	80.2
775x772	eixos+edificado	54.3	204.2	22.5	127.3

A Tabela 5.2.3 mostra alguns tempos medidos com um dispositivo real: um PDA HP Jornada.

Tabela 5.2.3 Tempos de de interpretação num dispositivo real: HP Jornada

Dimensão do mapa (m)	Camadas	Tempos (segundos)	
		Interpretação	Rendering
306x305	eixos+edificado	17.0	3.0
816x814	eixos+edificado	50.0	9.0

²Este tempo inclui também o tempo de ligação ao servidor, ou seja, é medido desde o momento em que é feito o pedido ao servidor até a interpretação do documento estar concluída.

5.2.1 Análise dos Resultados

Os valores apresentados indicam que o sistema apenas responde de forma aceitável para mapas muito pequenos. Quando a área abrangida pelo mapa é grande, ou quando escolhemos visualizar várias camadas, o tempo de espera aumenta consideravelmente tornando o sistema muito pouco usável.

Basicamente, existem dois gargalos no sistema:

- A transformação através de XSLT. Quando o documento GML se torna demasiado grande, a transformação para SVG torna-se demasiado lenta. Existem vários factores que contribuem para isto. A transformação é feita através de XSLT em Java usando a API JAXP. No nosso caso, é usado o processador de XML Saxon. A transformação através de XSLT requer a criação das árvores DOM tanto do documento a transformar como do documento XSLT. À medida que o documento a transformar aumenta, também aumenta o tempo de interpretação do documento e da criação da respectiva DOM. Para além disso, as árvores DOM requerem muita memória, o que, no nosso caso vai obrigar ao uso de memória virtual durante a transformação e conseqüente degradação do desempenho.
- A interpretação e *renderização* do SVG no dispositivo. Uma vez que estamos a lidar com dispositivos muito limitados a nível de processamento, é natural que o desempenho na interpretação e *renderização* dos documentos SVG seja fraco.

Fica também claro, pelos resultados apresentados nas tabelas 4.5.3 e 5.2.3, que há diferenças muito grandes a nível de processamento entre dispositivos diferentes. Apesar de termos usado um emulador no caso do telemóvel, ficamos, mesmo assim, com uma ideia da velocidade de processamento que podemos encontrar nalguns dispositivos reais. O artigo *J2ME devices: Real-world performance* [YRA02] contém *benchmarks* efectuados sobre alguns dispositivos MIDP, incluindo o WTK para comparação. Os resultados desses *benchmarks* indicam que, em quase todos os aspectos testados (e.g. velocidade das primitivas gráficas, comunicação, etc), o WTK é mais rápido do que os outros dispositivos (telemóveis) testados. Comparando os valores das tabelas 4.5.3 e 5.2.3 verificamos que o PDA é cerca de 4 vezes mais rápido a interpretar o SVG do que o WTK, mesmo com um mapa ligeiramente maior.

Capítulo 6

Conclusões e Trabalho Futuro

O objectivo principal deste projecto era obter uma resposta à pergunta:

Será viável desenvolver um sistema móvel de visualização de informação geográfica recorrendo a normas e formatos abertos?

Os resultados obtidos indicam que sim, mas com algumas limitações.

As principais limitações do sistema estão relacionadas com a quantidade de informação que o cliente consegue processar. Estas limitações são de duas ordens: a primeira diz respeito à memória necessária para visualizar um determinado mapa; a segunda relaciona-se com o tempo necessário para processar um mapa. Obviamente que estas são limitações de alguns dispositivos actuais. É de esperar que os próximos dispositivos melhorem o desempenho do sistema no que diz respeito a estas restrições.

Uma outra limitação do sistema prende-se com o aspecto gráfico. As APIs gráficas do MIDP são muito básicas pelo que apenas se conseguem produzir mapas muito simples graficamente. Uma possível forma de resolver este problema seria recorrer às APIs de alguns fabricantes de telemóveis, que disponibilizam funções gráficas um pouco mais completas. A desvantagem seria a redução da portabilidade da aplicação.

As limitações apontadas nos parágrafos anteriores dizem respeito ao cliente, no entanto, existem também algumas limitações, no nosso sistema, do lado do servidor. Estas limitações referem-se à transformação de GML para SVG. Contudo, uma vez que as opções que podemos tomar para a implementação do servidor são mais alargadas, estas limitações não causam muita preocupação. Seria perfeitamente possível, por exemplo, implementar a transformação de GML para SVG sem usar XSLT; a transformação pode ser feita usando *Perl*, ou mesmo Java, directamente sobre o documento GML. Obviamente, esta solução diminuiria a flexibilidade do sistema, mas poderia resolver o problema da eficiência da transformação.

6.1 Melhorias ao Sistema Desenvolvido

O sistema desenvolvido pode ser melhorado de várias formas:

Interface/Usabilidade Em termos de interface da aplicação cliente existem alguns aspectos que podem ser melhorados. Por exemplo, enquanto o mapa está a ser carregado não é

possível, neste momento, interromper o carregamento. Um melhoramento possível seria a introdução dessa possibilidade.

Transformação GML para SVG Como já foi referido, a transformação de GML para SVG torna-se demasiado lenta, quanto a área pedida é grande. Um possível melhoramento seria implementar a transformação de uma forma diferente, sem usar XSLT, de forma torná-la mais eficiente. Outra alternativa seria tentar otimizar algumas das funções do XSLT de forma a melhorar o tempo de execução. Esta optimização pode ser feita de duas formas: melhorando o código actual da folha de transformação; ou implementando o mesmo código, de forma mais eficiente, numa outra linguagem suportada pelo processador de XSLT e usado extensões do XSLT para invocar esse código.

6.2 Novas Funcionalidades

O desenvolvimento de um sistema como o m-GIS abre a porta a várias possíveis expansões ao nível das funcionalidades oferecidas pelo sistema. As novas funcionalidades apresentadas aqui são basicamente serviços baseados na localização, ou seja, funcionalidades que requerem o conhecimento da posição do utilizador. Estes serviços poderão ser implementados facilmente na tecnologia J2ME desde que o pacote opcional *Location API for J2ME (JSR-179)* seja implementado pelo dispositivo.

Localização actual Uma funcionalidade interessante num sistema deste género seria a possibilidade de o sistema exibir o mapa da zona onde o utilizador se encontra no momento. Um serviço deste tipo tem interesse, por exemplo, no caso de pessoas de visita a cidades desconhecidas.

Localização actual síncrona Uma funcionalidade muito parecida com a anterior seria a possibilidade de o sistema actualizar periodicamente o mapa exibido de forma a mostrar sempre o mapa da área onde o utilizador se encontra. Isto teria mais interesse para utilizadores em viagem.

Serviço do género “Aqui Perto” No início de 2001, a TMN lançou o serviço “Aqui Perto”, que permite encontrar uma farmácia de serviço, um restaurante ou um posto de gasolina perto de onde o utilizador se encontra. A informação resultante resume-se aos contactos dos estabelecimentos encontrados na zona do utilizador.

Um serviço semelhante pode ser expandido de forma a retornar o mapa da área, possivelmente com a indicação dos estabelecimentos encontrados.

Existem ainda outras funcionalidade que podem ser implementadas mais facilmente uma vez que não recorrem ao conhecimento da posição do utilizador:

Pontos de Interesse Uma funcionalidade facilmente implementável seria a adição de uma lista de pontos de interesse na área visualizada pelo utilizador. Poderia ser acrescentado uma opção que listasse os pontos de interesse (possivelmente configurados pelo utilizador) da área visualizada, e.g., uma lista de monumentos, postos de bombeiros e de polícia, etc.

Pesquisa global Neste momento a pesquisa é efectuada sobre a área pedida pelo utilizador. Poder-se-ia implementar uma nova funcionalidade que permitisse ao utilizador, por exemplo, pesquisar por uma determinada rua em todo o concelho. O utilizador poderia, depois, visualizar o mapa da área em que situa a rua pretendida.

Referências

- [AAA⁺01] Ronald Ashri, Steve Atkinson, Danny Ayers, Marten Haglind, Bill Ray, Rob Machin, Nadia Nashi, Richard Taylor, and Chanoch Wiggers. *Professional Java Mobile Programming*. Wrox Press, 2001.
- [Al 03] Al Suttons' Software. ASXMLP, 2003. <http://www.alsutton.com/software/xmlparser>, consultado em Agosto de 2003.
- [Bra02] Neil Bradley. *The XML Companion*. Addison-Wesley, third edition, 2002.
- [dee03] deegree. Projecto deegree, 2003. <http://www.deegree.org>, consultado em Agosto de 2003.
- [dev03] J2ME Device (Issue) Database, 2003. <http://kissen.cs.uni-dortmund.de:8080/devicedb/index.html>, consultado em Agosto de 2003.
- [Dom02] Carlos Duarte Oliveira Domingues. Módulo Remoto para Aquisição de Informação Geográfica num PDA. Relatório do projecto, INESC Porto, 2002.
- [Enh03a] Enhydra.org. kSOAP Project, 2003. <http://ksoap.enhydra.org>, consultado em Agosto de 2003.
- [Enh03b] Enhydra.org. kXML Project, 2003. <http://kxml.enhydra.org>, consultado em Agosto de 2003.
- [ESR03] ESRI. ESRI - GIS and Mapping Software, 2003. <http://www.esri.com>, consultado em Agosto de 2003.
- [Gig02] Eric Giguere. J2ME Optional Packages. <http://wireless.java.sun.com/midp/articles/>, Dezembro 2002.
- [Gir03] Andrew Girow. TinyLine, 2003. <http://www.tinyline.com>, consultado em Agosto de 2003.
- [GJS96] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. Addison-Wesley, 1996.
- [HB97] Donald Hearn and M. Pauline Baker. *Computer Graphics*. Prentice-Hall, second edition, 1997.
- [IBM03] IBM. Web Services Tool Kit for Mobile Devices, 2003. <http://www.alphaworks.ibm.com/tech/wstkmd>, consultado em Agosto de 2003.

- [JSh03a] JShape. KShape, 2003. <http://www.jshape.com>, consultado em Agosto de 2003.
- [JSh03b] JShape. MShape, 2003. <http://www.jshape.com>, consultado em Agosto de 2003.
- [JSh03c] JShape. WShape, 2003. <http://www.jshape.com>, consultado em Agosto de 2003.
- [jT03] jScience Technologies. MathFP, 2003. <http://www.jscience.net>, consultado em Agosto de 2003.
- [Knu02] Jonathan Knudsen. Parsing XML in J2ME[tm]. <http://wireless.java.sun.com/midp/articles/parsingxml>, Março 2002.
- [Kur02] Budi Kurniawan. *Java for the Web with Servlets, JSP, and EJB*. New Riders, first edition, 2002.
- [LY97] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, 1997.
- [Mas03] OS MasterMap. Style and XML Examples, 2003. http://www.ordnancesurvey.co.uk/os/_mastermap/xml/, consultado em Agosto de 2003.
- [ME403] ME4SE. ME4SE, 2003. <http://www.me4se.org>, consultado em Agosto de 2003.
- [Mic03] Sun Microsystems. Personal Java, 2003. <http://java.sun.com/products/personaljava/>, consultado em Agosto de 2003.
- [Mon02] Anabela Soares Pinto Monteiro. Servidor de Mapas Vectoriais. Relatório do projecto, INESC Porto, 2002.
- [OCG01] OCG. Geography Markup Language (GML) 2.0. Especificação, OCG, Fevereiro 2001.
- [OCG02a] OCG. *Web Feature Service Implementation Specification*, 2002. <http://www.opengis.org>.
- [OCG02b] OCG. *Web Map Service Implementation Specification*, 2002. <http://www.opengis.org>.
- [OCG03] OCG. Consórcio *Open GIS*, 2003. <http://www.opengis.org>, consultado em Agosto de 2003.
- [Sof03] Wingfoot Software. Wingfoot SOAP, 2003. <http://www.wingfoot.com/index.jsp>, consultado em Agosto de 2003.
- [STK01] James Snell, Doug Tidwell, and Pavel Kulchenko. *Programming Web Services with SOAP*. O'Reilly, third edition, 2001.
- [Sun00a] Sun. Connected Limited Device Configuration - Version 1.0a. Especificação, Sun Microsystems, Maio 2000.

- [Sun00b] Sun. Mobile Information Device Profile 1.0a. Especificação, Sun Microsystems, Dezembro 2000.
- [Sun03] Sun. Connected Limited Device Configuration - Version 1.1. Especificação, Sun Microsystems, Março 2003.
- [Sys99] Inc. Adobe Systems. *PostScript(R) Language Reference*. Addison-Wesley, third edition, 1999.
- [W3C03a] W3C. Extensible Markup Language (XML), 2003. <http://www.w3.org/XML/>, consultado em Agosto de 2003.
- [W3C03b] W3C. SVG, 2003. <http://www.w3.org/Graphics/SVG/Overview.htm8>, consultado em Agosto de 2003.
- [xpa03] Xparse-J, 2003. <http://www.webreference.com/xml/tools/xparse-j.html>, consultado em Agosto de 2003.
- [YRA02] Wang Yi, C.J. Reddy, and Gavin Ang. J2me devices: Real-world performance. <http://www.javaworld.com/javaworld/jw-10-2002/jw-1025-j2mebenchmark.html>, Outubro 2002.

Glossário

C

CLDC *Connected Limited Device Configuration* — Uma das configurações (conjunto de APIs comuns a uma vasta gama de dispositivos) do J2ME. Esta configuração é orientada para dispositivos muito limitados, e.g., telemóveis e PDAs, com capacidade de ligação à rede, pág. 8.

D

DOM *Document Object Model* — O DOM é uma interface independente da plataforma e da linguagem de programação, que permite aos programas e *scripts* aceder e actualizar o conteúdo, estrutura e estilo de documentos, pág. 7.

G

GIS *Geographic Information System* [Ver SIG], pág. 5.

GML *Geography Markup Language* — Uma especificação desenvolvida pelo consórcio OpenGIS. O GML é uma linguagem XML, para o transporte e armazenamento de informação geográfica, incluindo tanto as propriedades espaciais como as não espaciais das características geográficas, pág. 3.

J

J2ME *Java 2 Micro Edition* — Uma tecnologia Java destinada a dispositivos com limitações de memória e de processamento quando comparados com os PCs de secretária, pág. 7.

JAXP *Java API for XML Processing* — Uma API que suporta processamento de documentos XML usando DOM, SAX e XSLT. O JAXP permite às aplicações interpretar e transformar documentos XML independentemente de um processador de XML particular, pág. 25.

JSR *Java Specification Request* — Os JSRs são as descrições actuais das propostas de especificação e especificações finais para a plataforma Java. Em qualquer momento, existem inúmeros JSRs a passar pelo processo de revisão e aprovação, pág. 11.

M

MIDP *Mobile Information Device Profile* — O único perfil (conjunto de APIs que aumentam as funcionalidades das configurações mas restringem a gama de dispositivos) para a CLDC, até ao momento. Este é o perfil implementado pelos telemóveis que executam aplicações Java, pág. 8.

O

OpenGIS *Open GIS Consortium* — O OGC é um consórcio internacional composto por empresas, agências governamentais e universidades que participam para desenvolver especificações, públicas, relacionadas com sistemas geográficos, pág. 3.

S

SIG Sistema de Informação Geográfica — Um Sistema de Informação Geográfica é um sistema de informação que armazena, manipula, analisa e devolve informação geo-referenciada e informação geo-espacial, de forma a auxiliar a tomada de decisão referente ao uso e gestão dos recursos naturais, ambiente transportes, instalações urbanas, etc, pág. 5.

SVG

Scalable Vector Graphics — é um formato vectorial de imagens, especificado em XML. Ao contrário dos formatos *raster* (que consistem em mapas de bits), um formato vectorial descreve uma imagem através fórmulas geométricas. Uma das vantagens dos formatos vectoriais é a possibilidade de redimensionar a imagem sem perder detalhe, pág. 7.

W

WFS *Web Feature Service/Server* — O WFS especifica o serviço de *Web Features*, ou seja, define a interface de um sistema que produz informação geográfica no formato GML, pág. 11.

WMS

Web Map Service/Server — O WMS especifica o serviço de *Web Map*, ou seja, especifica a interface, *Web*, de um sistema que produz mapas (representações visuais de informação geográfica). A especificação define os tipos de pedidos e respostas que um serviço deste género deve suportar, pág. 11.

X

XML *eXtensible Markup Language* — uma especificação desenvolvida pelo W3C. É, basicamente, uma linguagem que permite o armazenamento de informação de forma estruturada e acessível. O XML permite a criação de elementos personalizados, facilitando a definição, transmissão, validação e interpretação de dados entre aplicações e organizações, pág. 6.

XSLT *XSL Transformations* — XSLT é uma forma de especificar uma transformação a aplicar a um documento XML. A especificação da transformação é, ela própria feita em XML, pág. 7.

Apêndice A

J2ME

Uma vez que a tecnologia J2ME, mais especificamente o perfil MIDP, está na base da aplicação cliente desenvolvida neste projecto, este anexo apresenta uma visão mais aprofundada dessa tecnologia.

A.1 Introdução

O J2ME (*Java 2 Micro Edition*) é, à semelhança do J2SE (*Java 2 Standard Edition*) ou do J2EE (*Java 2 Enterprise Edition*), uma tecnologia Java da Sun.

O J2ME nasceu da necessidade de adaptar a tecnologia Java aos dispositivos móveis com sérias limitações de recursos quando comparados com os PCs de secretária.

É um ambiente Java direccionado a uma vasta gama de dispositivos móveis, que vai desde *Smart Cards* até *Set-top Boxes*, passando por telemóveis e PDAs (*Personal Digital Assistants*). Consiste num conjunto de especificações organizadas em camadas que permitem abranger um largo leque de dispositivos e tecnologias.

Basicamente, a arquitectura do J2ME está dividida em três camadas como se pode ver na Figura A.1.1: máquina virtual, configurações e perfis.

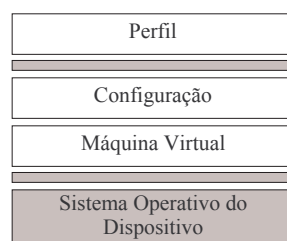


Figura A.1.1: Arquitectura de um ambiente J2ME

A *máquina virtual* está colocada directamente acima do sistema operativo do dispositivo. É a máquina virtual que define quais são as limitações dos programas que podem correr no dispositivos.

Uma *configuração* é um conjunto de bibliotecas básicas que estão disponíveis para o programador. A configuração define também o nível de funcionalidades e serviços que têm de ser

oferecidos pela máquina virtual. Uma configuração é especificada para uma classe horizontal de dispositivos, i.e., uma gama de dispositivos, com diferentes aplicações, mas que partilham algumas características. Por exemplo, a classe dos dispositivos de informação com comunicação *wireless* abrange vários tipos de dispositivos, desde telemóveis, PDAs, *paggers*, etc.

Finalmente, o *perfil* define um conjunto de bibliotecas específicas para uma classe vertical de dispositivos. Seguindo o exemplo anterior, poderíamos ter um perfil para telemóveis, outro para PDAs, etc. As bibliotecas definidas por um perfil são mais específicas do que as definidas pela configuração uma vez que a gama de dispositivos alvo também é menor. Um perfil é sempre especificado para uma determinada configuração, mas uma configuração pode suportar vários perfis.

Neste momento existem apenas duas configurações definidas:

Connected Limited Device Configuration - CLDC Esta configuração é usada em dispositivos muito limitados, e.g., telemóveis e PDAs, com capacidade de ligação à rede.

Connected Device Configuration - CDC Usada em dispositivos com mais capacidade e com melhor conectividade à rede.

A configuração que nos interessa é a CLDC, uma vez que é esta a configuração suportada pelos telemóveis.

O único *perfil* definido até este momento para a CLDC é o *Mobile Information Device Profile* — MIDP. O MIDP define alguns requisitos mínimos para o *hardware*, por exemplo, o ecrã deve ter pelo menos 96x54 pixels, o dispositivo deve ter pelo menos 32Kb de memória volátil para o *heap* do *Java runtime*, etc.

A configuração CLDC em conjunto com o perfil MIDP constituem o ambiente Java que se encontra em grande parte dos telemóveis disponíveis hoje em dia.

A.2 *Connected Limited Device Configuration (CLDC)*

Uma configuração da plataforma J2ME especifica um subconjunto da linguagem de programação Java, um subconjunto das funcionalidades da máquina virtual Java da configuração, características de segurança e comunicação e também as bibliotecas nucleares da plataforma.

Isto significa que uma determinada configuração J2ME determina quais as características mínimas da máquina virtual (e.g., se suporta números de vírgula flutuante ou não, que tipo de segurança oferece, etc), quais as bibliotecas básicas e quais os atributos da linguagem Java que o programador terá ao seu dispor.

Obviamente que qualquer configuração J2ME restringe também as características dos dispositivos que a suporta.

A.2.1 **Requisitos Mínimos**

No caso da CLDC 1.0¹ os limites mínimos do *hardware* são os seguintes:

¹A versão mais recente da CLDC, na altura em que este relatório foi elaborada, era a versão 1.1. No entanto, uma vez que ainda não existiam implementações desta versão, vamos concentrar-mo-nos na versão 1.0.

- Pelo menos 128 kilobytes de memória não volátil está disponível para a máquina virtual e as bibliotecas CLDC.
- Pelo menos 32 kilobytes de memória volátil está disponível para o *runtime* da máquina virtual (por exemplo para o *heap*).

Estes requisitos são requisitos *mínimos*, obviamente que os dispositivos reais podem ter mais recursos do que os mínimos.

Em relação aos requisitos de *software*, a CLDC assume apenas que um sistema operativo mínimo, ou núcleo (*kernel*), está disponível para gerir o *hardware*. O sistema operativo não necessita de garantir nenhum escalonamento em tempo real nem suportar diferentes espaços de endereçamento nem processos.

A.2.2 Diferenças da CLDC para um ambiente Java “normal”

O objectivo geral de uma máquina virtual que esteja conforme a especificação CLDC [Sun00a] é aproximar-se tanto quanto possível de um ambiente Java “normal”, i.e., um ambiente que está conforme a Especificação da Linguagem Java (*Java Language Specification* [GJS96]) e a Especificação da Máquina Virtual Java (*Java Virtual Machine Specification* [LY97]), dentro das limitações dos dispositivos alvo da CLDC. Assim é mais prático descrever a especificação CLDC em termos das diferenças para esse ambiente Java.

Em termos das características da máquina virtual as principais diferenças são:

Não suporta números de vírgula flutuante² O suporte para números de vírgula flutuante foi removido porque a maioria dos dispositivos alvo não têm suporte para vírgula flutuante por *hardware* e o custo do suporte de vírgula flutuante por *software* foi considerado muito alto. Isto significa que a máquina virtual não permite o uso de literais, operações, tipos ou valores de vírgula flutuante, e.g.,

```
float x = 0.1f;
```

não é permitido.

Não suporta finalização de objectos As bibliotecas da configuração CLDC não incluem o método *finalize()* da classe *Object* pelo que a máquina virtual não necessita de suportar finalização de objectos. As aplicações desenvolvidas não podem requerer esta funcionalidade.

Limitações ao nível do tratamento de erros e excepções A máquina virtual da CLDC suporta excepções, mas não excepções assíncronas³. As classes de erro incluídas na CLDC também são mais limitadas. O conjunto de classes de erros e excepções da CLDC é apresentado mais à frente.

Java Native Interface (JNI) A máquina virtual não implementa o JNI. O JNI foi excluído por razões de segurança e por limitações de memória.

²A versão 1.1 da CLDC já inclui suporte para vírgula flutuante.

³Excepções assíncronas são excepções em que o ponto do programa em que ocorrem não é conhecido. Um exemplo de excepção assíncrona é a *InternalError* causada por um erro na máquina virtual. Para mais informação ver [GJS96].

Reflexão (Reflection) A máquina virtual não suporta reflexão, i.e., não permite a um programa inspeccionar o número e o conteúdo de classes, métodos, campos, *threads*, pilhas de execução e outras estruturas da máquina virtual. Como consequência desta limitação a máquina virtual não suporta RMI (*Remote Method Invocation*), JVMDI (*Debugger Interface*), JVMPI (*Profiler Interface*) e outras características avançadas que necessitem de reflexão.

Não suporta *class loaders* definidos pelo utilizador Esta restrição foi imposta por questões de segurança. O carregamento de classes é feito pelo *class loader* da máquina virtual e não pode ser substituído ou reconfigurado.

Não suporta grupos de *threads* nem *daemon threads* A máquina virtual suporta *threading* mas não suporta grupos de *threads* nem *daemon threads*. O suporte para grupos de *threads* tem de ser feito a nível da aplicação, caso estas necessitem desta funcionalidade.

Referências fracas (*Weak References*)⁴ A máquina virtual CLDC não suporta referências fracas.

Verificação de classes feita de forma alternativa A verificação das classes pode ser feita de duas formas:

- Usando o método usual de verificação das classes definido na Especificação da Máquina Virtual.
- Usando um método mais eficiente que consiste em realizar parte da verificação aquando do desenvolvimento da aplicação. A máquina virtual do dispositivo apenas faz parte verificação.

A biblioteca de classes fornecida pela CLDC divide-se em duas partes: classes derivadas do J2SE e classes específicas da CLDC.

As classes derivadas do J2SE são:

Classes de Sistema

`java.lang.Object`, `Class`, `Runtime`,
`System`, `Thread`, `Runnable` (interface),
`String`, `StringBuffer`, `Throwable`

Classes de Tipos de Dados

`java.lang.Boolean`, `Byte`, `Short`, `Integer`, `Long`, `Character`

Classes de Coleções

`java.util.Vector`, `Stack`, `Hashtable`, `Enumeration` (interface)

Classes de *Input/Output*

`java.io.InputStream`, `OutputStream`,
`ByteArrayInputStream`, `ByteArrayOutputStream`,
`DataInput` (interface), `DataOutput` (interface),
`DataInputStream`, `DataOutputStream`,
`Reader`, `Writer`,
`InputStreamReader`, `OutputStreamWriter`,
`PrintStream`

Classes de Calendário e Tempo

`java.util.Calendar`, `Date`, `TimeZone`

⁴A versão CLDC 1.1 já suporta referências fracas.

Classes Adicionais `java.util.Random`, `java.lang.Math`

Classes de Exceções

`java.lang.Exception`, `ArithmeticException`,
`ArrayIndexOutOfBoundsException`, `ArrayStoreException`,
`ClassCastException`, `ClassNotFoundException`,
`IllegalAccessException`, `IllegalArgumentException`,
`IllegalMonitorStateException`, `IllegalThreadStateException`,
`IndexOutOfBoundsException`, `InstantiationException`,
`InterruptedException`, `NegativeArraySizeException`,
`NullPointerException`, `NumberFormatException`,
`RuntimeException`, `SecurityException`,
`StringIndexOutOfBoundsException`,
`java.util.EmptyStackException`, `NoSuchElementException`,
`java.io.EOFException`, `InterruptedIOException`,
`IOException`, `UnsupportedEncodingException`,
`UTFDataFormatException`,

Classes de Erros

`java.lang.Error` `OutOfMemoryError` `VirtualMachineError`

Para além das classes derivadas do J2SE existem também algumas específicas da CLDC. Estas classes fazem toda a parte do pacote `javax.microedition.io` e constituem o chamado *Generic Connection framework*:

- *Connection* (Interface)
- *ContentConnection* (Interface)
- *Datagram* (Interface)
- *DatagramConnection* (Interface)
- *InputConnection* (Interface)
- *OutputConnection* (Interface)
- *StreamConnection* (Interface)
- *StreamConnectionNotifier* (Interface)
- `Connector`
- `ConnectionNotFoundException`

A configuração *Connected Limited Device* é a base para um ou vários perfis. Um perfil define um conjunto de bibliotecas adicionais que fazem sentido num determinado conjunto vertical de dispositivos, e.g., telemóveis. Até à data apenas um perfil está definido para a CLDC: o perfil MIDP — *Mobile Information Device Profile*.

A.3 Mobile Information Device Profile (MIDP)

Um perfil estende a API de uma configuração acrescentando-lhe funcionalidades que fazem sentido numa determinada gama de dispositivos, e.g., telemóveis.

O MIDP (*Mobile Information Device Profile*) é um (e até agora, o único) perfil definido para a CLDC. O MIDP está agora na versão 2.0 embora a grande maioria dos dispositivos existentes implementem apenas a versão 1.0. Este Anexo foca principalmente a versão 1.0.

A arquitectura geral de um dispositivo MIDP é indicada na Figura A.3.1. Todos os dispositivos conformes com a especificação MIDP implementam as classes da configuração CLDC e as classes do perfil MIDP. Estas classes tentam providenciar uma base comum de desenvolvimento de aplicações, mas obviamente não cobrem todas as funcionalidades de dispositivos particulares. Para colmatar estas restrições, alguns fabricantes incluem bibliotecas Java próprias de forma a que as aplicações desenvolvidas possam tirar partido de funcionalidades próprias dos seus dispositivos. As aplicações desenvolvidas usando estas bibliotecas são específicas de um determinado dispositivo (ou dispositivos de um fabricante), não podendo ser, por isso, executadas noutros dispositivos.

Por outro lado as aplicações escritas para MIDP em geral podem correr em qualquer dispositivo que siga a especificação MIDP.

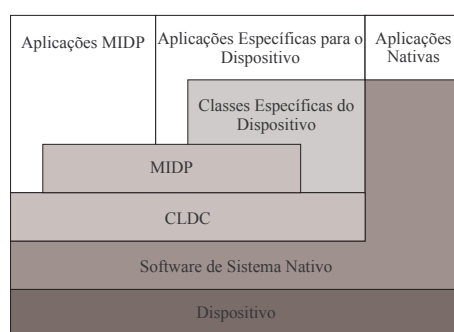


Figura A.3.1: Arquitectura de um dispositivo MIDP (adaptado de [Sun00b])

O perfil MIDP foi desenvolvido para dispositivos MID (*Mobile Information Device*). Segundo a especificação MIDP1.0 [Sun00b] um dispositivo MID tem as seguintes características mínimas de *hardware*:

Ecrã

- Tamanho: 96x54
- Profundidade de cor: 1-bit
- Aspecto do pixel (*aspect ratio*): aproximadamente 1:1

Entrada

- Um ou mais dos mecanismos de entrada seguintes: “teclado de uma mão”⁵, “teclado de duas mãos”⁶ ou ecrã táctil.

⁵“Teclado de uma mão” é um termo usado para descrever os teclados de telefone ITU-T.

⁶“Teclado de duas mãos” é um termo usado para descrever os teclados QWERTY de computador.

Memória⁷

- 128 kilobytes de memória não volátil para os componentes MIDP
- 8 kilobytes de memória não volátil para dados persistentes criados pelas aplicações.
- 32 kilobytes de memória volátil para o *runtime* do Java (e.g., *heap* do Java)

Comunicação

- Comunicação *wireless*, em dois sentidos, possivelmente intermitente e com largura de banda limitada.

Em relação ao *software* o MIDP assume apenas funcionalidades mínimas uma vez que ao nível dos dispositivos MID há uma grande variedade de arquitecturas de software de sistema. Os requisitos mínimos são os seguintes:

- Um mecanismo para ler e escrever em memória não-volátil.
- Acesso de leitura e escrita ao sistema de comunicação *wireless* do dispositivo.
- Um mecanismo que forneça uma base de tempo usada em *timestamping*.
- Capacidade mínima para escrever gráficos do tipo mapas de bits no ecrã.
- Um mecanismo para capturar *input* do utilizador.
- Um mecanismo para gerir o ciclo de vida das aplicações do dispositivo.

A.3.1 Bibliotecas do MIDP

O MIDP oferece ao programador algumas bibliotecas básicas para o desenvolvimento de aplicações. Estas bibliotecas abrangem as seguintes áreas:

Aplicação Define o ciclo de vida da aplicação e a forma como é controlada pelo dispositivo.

Interface Gráfica Disponibiliza funções para construir interfaces gráficas e para obter o *input* do utilizador.

Armazenamento persistente Não existe o conceito de ficheiro em MIDP, no entanto existem funções para ler e escrever dados num sistema de armazenamento permanente.

Comunicação O MIDP disponibiliza funções para aceder ao sistema de comunicação do dispositivo.

Temporizadores Existem funções para escalonar *threads* para execução futura em *background*.

⁷Estes requisitos de memória são para os componentes MIDP apenas. Os requisitos para a CLDC e outros requisitos de memória do sistema não são considerados.

A.3.2 MIDlets

As aplicações escritas para o perfil MIDP são designadas por *MIDlets*.

Uma MIDlet consiste numa classe que estende a classe `MIDlet` e nas outras classes necessárias à aplicação. As classes que compõem a MIDlet são empacotadas num ficheiro JAR e a instalação da MIDlet no dispositivo consiste em descarregar o JAR para o dispositivo. A transferência do JAR para o dispositivo pode ser feita de várias formas, dependendo do dispositivo. Alguns podem permitir a transferência por infravermelhos, *Bluetooth*, por ligação directa ao computador, etc. Existe, no entanto, uma forma de transferência que deve ser suportada por todos os dispositivos MIDP: *Over The Air*⁸ (OTA). A transferência OTA é, basicamente, a transferência por HTTP dos ficheiros JAR e JAD (descrito mais à frente). Desta forma é possível colocar a MIDlet num servidor HTTP e fazer a instalação em qualquer dispositivo através do protocolo HTTP.

Várias MIDlets podem ser empacotadas num ficheiro JAR; isto é designado por *MIDlet Suite*. O ficheiro JAR contém um ficheiro de manifesto que descreve a *MIDlet Suite* e cada uma das MIDlets presentes.

Para além do ficheiro de manifesto, que se encontra no ficheiro JAR, existe outro ficheiro, que não é incluído no JAR, usado para descrever as MIDlets. Este ficheiro tem a extensão `.jad` (*Java application descriptor*) e serve basicamente para descrever a *MIDlet Suite*, e.g., tamanho do ficheiro JAR, URL do ficheiro JAR, nome da *MIDlet Suite*, fornecedor, versão, e para descrever cada uma das MIDlets presentes, e.g., nome, ícone. O objectivo principal do ficheiro JAD é permitir ao dispositivo decidir se tem capacidade para executar a aplicação antes de a descarregar. Se a *MIDlet Suite* for demasiado grande para a capacidade de memória do dispositivo não faz sentido descarregar o ficheiro JAR.

Uma MIDlet tem um ciclo de vida semelhante às Applets. Há três estados possíveis no ciclo de vida de uma MIDlet:

inactiva A instância da MIDlet foi construída e está inactiva.

activa A MIDlet está activa.

destruída A MIDlet foi terminada.

Depois de construída a MIDlet pode alternar entre o estado *activa* e *inactiva* várias vezes. O estado da MIDlet é gerido pelo *software* de gestão de aplicações (*Application Management Software* — AMS) do dispositivo.

A.4 Pacotes opcionais

Um pacote opcional é um conjunto de bibliotecas que, ao contrário de um perfil, não define um ambiente aplicacional completo. Um pacote opcional estende o ambiente de execução de forma a suportar capacidades que não são suficientemente universais para fazerem parte de um perfil. Um pacote opcional é sempre usado em conjunção com uma determinada configuração, mas pode estar disponível para vários perfis.

⁸A versão 1.0 do MIDP apenas recomendava a implementação deste método de instalação de aplicações. A versão 2.0, no entanto, obriga a que um dispositivo MIDP suporte esta funcionalidade.

À semelhança das configurações e dos perfis, é o fornecedor dos dispositivos que decide que pacotes opcionais irão estar disponíveis ao programador.

Existem vários pacotes opcionais em fase de definição através do *Java Community Process*, dos quais destacamos dois:

JSR 120: WMA – *Wireless Messaging API* O WMA estende o *Generic Connection Framework* permitindo às aplicações enviar e receber mensagens usando o serviço *Short Message Service* (SMS).

JSR 135: MMAPI – *Mobile Media API* O MMAPI permite a uma aplicação capturar e reproduzir conteúdo multimédia. Algumas classes deste pacote opcional foram mesmo incluídas na versão 2.0 do MIDP.

Apêndice B

GML

Este anexo descreve a linguagem GML em mais pormenor do que o que foi apresentado no relatório, já que esta linguagem é a base da informação geográfica utilizada neste projecto.

O GML (*Geography Markup Language*) é uma linguagem XML, para o transporte e armazenamento de informação geográfica, incluindo tanto as propriedades espaciais com as não espaciais dos objectos geográficos ([OCG01]).

O GML é uma especificação desenvolvida pelo consórcio internacional OpenGIS. O grande objectivo do GML é fornecer uma plataforma neutra e aberta para a definição de objectos e esquemas geo-espaciais. O GML não é uma linguagem rígida mas antes um sistema que suporta a definição de linguagens de descrição geográfica. Posto muito simplesmente, o GML consiste num esquema (*schema*) XML que pode ser estendido de forma a se adaptar a situações específicas, mas mantendo sempre uma base comum.

B.1 Visão Geral

O GML foi desenvolvido tendo em vista vários objectivos, de acordo com a especificação [OCG01]:

- Providenciar uma forma de codificar informação espacial tanto para transporte de dados como para armazenamento.
- Ser suficientemente extensível de forma a suportar uma variedade de tarefas, desde a visualização até à análise.
- Estabelecer a base para o *Internet GIS* de uma forma modular e incremental.
- Permitir a codificação eficiente de geometria geo-espacial.
- Fornecer uma codificação de informação espacial e relações espaciais fáceis de compreender.
- Ser capaz de separar conteúdo espacial e não espacial da apresentação de dados (gráfica, ou não).
- Permitir a fácil integração de dados espaciais e não espaciais.

- Permitir a ligação entre elementos espaciais (geométricos) e outro tipo de elementos espaciais ou não espaciais.
- Fornecer um conjunto de objectos de modelação geográfica de forma a permitir a interoperabilidade de aplicações desenvolvidas independentemente.

O GML está desenhado de forma a suportar a interoperabilidade e fá-lo através da definição de elementos geométricos básicos (todos os sistemas que suportam GML usam os mesmos elementos geométricos), de um modelo de dados comum e um mecanismo para criação e partilha de esquemas (*schemas*) aplicativos. A maior parte das comunidades de informação aumentam a sua interoperabilidade através da publicação dos seus esquemas.

O GML foi desenvolvido tendo em conta o princípio da separação entre conteúdo e apresentação. O GML fornece elementos para a codificação de dados de objectos geográficos sem se preocupar como esses dados serão apresentados. Uma vez que o GML é uma linguagem XML, é facilmente transformável num formato mais adequado à apresentação, seja ela gráfica, sonora, textual, etc.

B.2 Modelo de Objectos

O GML é uma codificação em XML para objectos geográficos. Um objecto geográfico em GML é, essencialmente, uma lista de propriedades, em algumas dessas propriedades podem ser geo-espaciais, descrevendo a forma e posição do objecto. Um objecto geográfico pode ser visto como um objecto na terminologia de modelação por objectos. Um objecto geográfico tem um tipo, que pode ser visto como a classe, na mesma terminologia de modelação por objectos. A “classe” descreve quais as propriedades que um objecto desse tipo deve ter. As propriedades podem ser modeladas em UML como associações, ou como atributos, da classe. Os valores das propriedades são, eles próprios, instâncias de classes, ou tipos de objectos definidos.

Propriedades Geométricas

Em geral, a definição de propriedades dos objectos é feita nos esquemas da aplicação. No entanto, o GML define alguns elementos de propriedades geométricas para associar estas geometrias com os objectos geográficos.

Os nomes formais e descritivos para as propriedades básicas são listados na Tabela B.2.1. Estes nomes aparecem no esquema *Feature* para designar propriedades geométricas comuns. A semântica precisa destas propriedades não é especificada.

Esquemas Aplicacionais

O GML fornece três esquemas XML base: *feature.xsd* que define o modelo objecto-propriedade geral, *geomtry.xsd* que inclui componentes geométricos e *xlinks.xsd* que fornece os atributos XLink usados para implementar a funcionalidade de ligações (*linking*).

Estes três documentos de esquema fornecem tipos básicos e estruturas que podem ser usadas pelos esquemas aplicativos. Um esquema aplicativo declara o tipos de objectos geográficos

Tabela B.2.1 Propriedades geométricas básicas (retirado de [OCG01])

Nome formal	Nome descritivo	Tipo geométrico
boundedBy	-	Box
pointProperty	location, position, centerOf	Point
lineStringProperty	centerLineOf, edgeOf	LineString
polygonProperty	extentOf, coverage	Polygon
geometryProperty	-	any
multiPointProperty	multiLocation, multiPosition, multiCenterOf	MultiPoint
multiLineStringProperty	multiCenterLineOf, multiEdgeOf	MultiLineString
multiPolygonProperty	multiExtentOf, multiCoverage	MultiPolygon
multiGeometryProperty	-	MultiGeometry

e tipos de propriedades que interessam para um domínio particular, usando componentes do GML de uma forma normalizada.

B.3 Codificação de GML

As Figuras B.3.1 e B.3.2 mostram os diagramas UML representativos dos esquemas de geometria e de objectos geográficos, respectivamente. Estes são os esquemas mais importantes ao codificar GML. De seguida são dados alguns exemplos de codificação de objectos geográficos em GML. Todos estes exemplos são retirados de [OCG01].

Codificação de Geometria

O GML fornece as seguintes classes geométricas:

- Point
- LineString
- LinearRing
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon
- MultiGeometry

Para além destas classes existem ainda o elemento `<Box>` para definir extensões e os elementos `<coord>` e `<coordinates>` para definir coordenadas.

A seguir são dados exemplos da utilização de algumas destas classes.

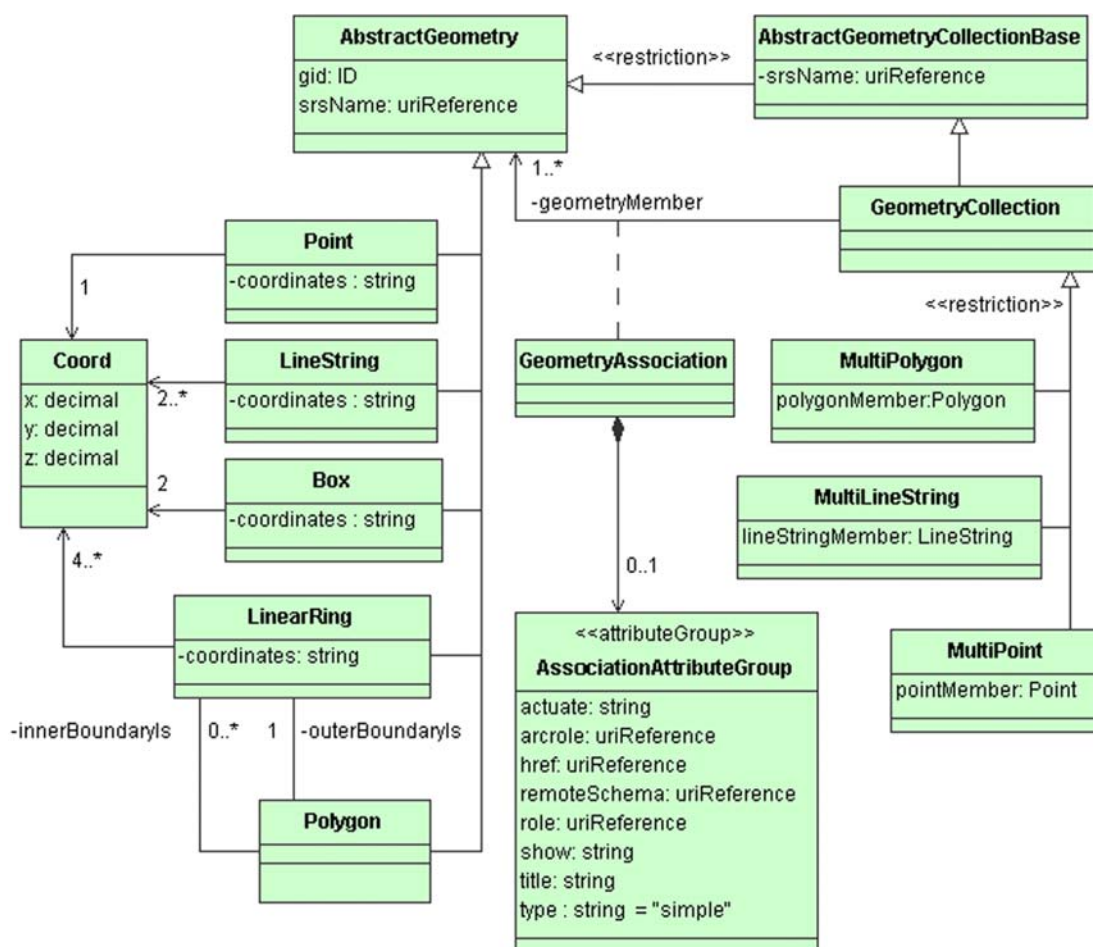


Figura B.3.1: Esquema geométrico (retirado de [OCG01])

O elemento `<Point>` é usado para codificar instâncias da classe `Point`. As coordenadas do ponto podem ser dadas através de um único elemento `<coord>` ou de um elemento `<coordinates>` com um único tuplo. Uma codificação possível para um ponto é a seguinte:

```
<Point gid="P1" srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
  <coord><X>56.1</X><Y>0.45</Y></coord>
</Point>
```

Um `<LineString>` representa um caminho (*path*), i.e., conjunto de pontos ligados por segmentos de recta. O caminho pode ser fechado se os primeiro e último pontos coincidirem. O exemplo seguinte mostra uma codificação de um `LineString`:

```
<LineString srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
  <coord><X>0.0</X><Y>0.0</Y></coord>
  <coord><X>20.0</X><Y>35.0</Y></coord>
  <coord><X>100.0</X><Y>100.0</Y></coord>
</LineString>
```

O elemento `<LinearRing>` é muito semelhante ao `<LineString>` excepto o facto de ser uma forma fechada, i.e., o último ponto tem de coincidir com o primeiro. Para além disso

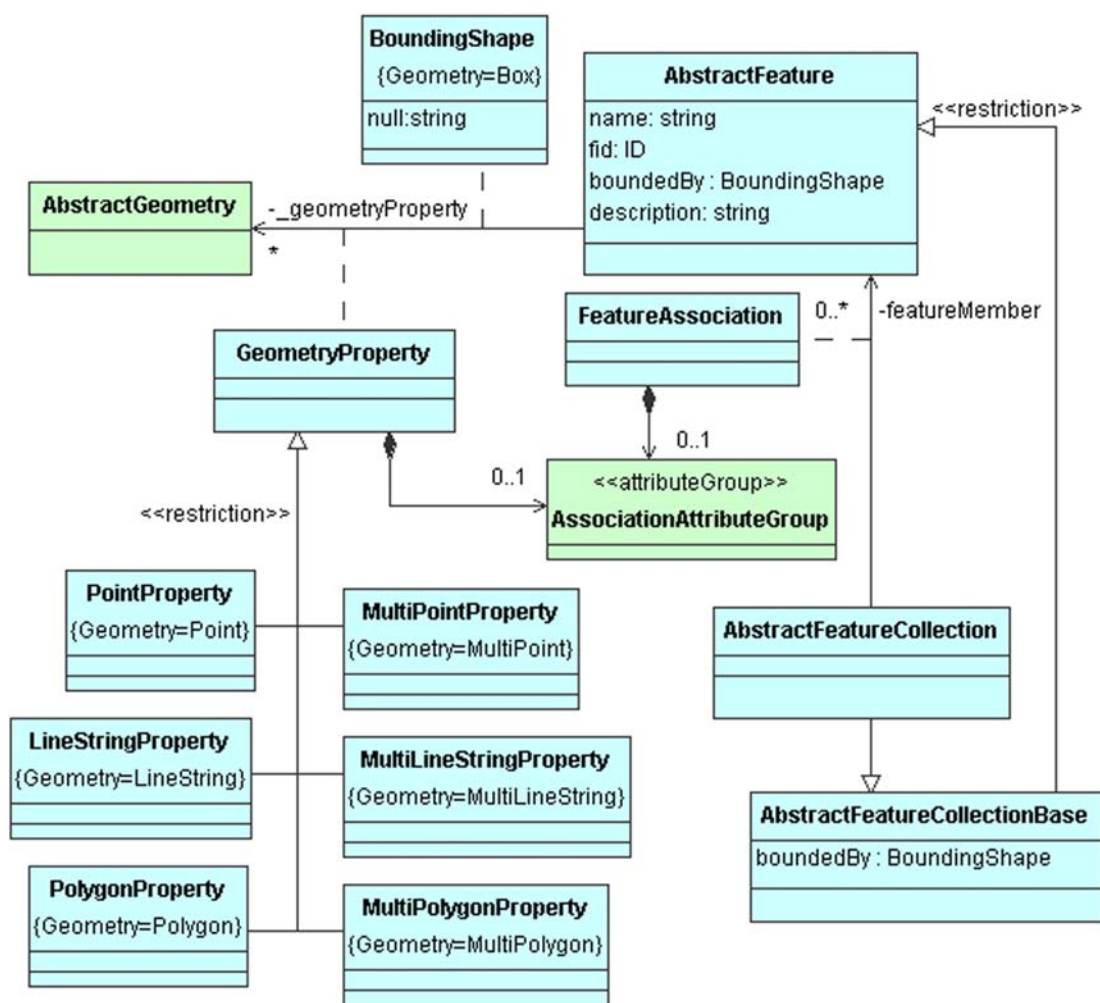


Figura B.3.2: Esquema de objectos geográficos (retirado de [OCG01])

precisa de pelo menos quatro coordenadas, três para definir o anel e a última que tem de ser coincidente com a primeira. Uma *LineString* é usada para construir polígonos.

Um *Polygon* é uma superfície. É definido através de *LinearRings*, uma, obrigatória, define a fronteira exterior, as outras definem as fronteiras interiores, i.e., “buracos” na superfície. Aqui está um exemplo de codificação de um *Polygon*:

```

<Polygon gid="_98217" srsName="http://www.opengis.net/gml/srs/epsg.xml#4326"
  <outerBoundaryIs>
    <LinearRing>
      <coordinates>
        0.0,0.0 100.0,0.0 100.0,100.0 0.0,100.0 0.0,0.0
      </coordinates>
    </LinearRing>
  </outerBoundaryIs>
  <innerBoundaryIs>
    <LinearRing>
      <coordinates>
        10.0,10.0 10.0,40.0 40.0,40.0 40.0,10.0 10.0,10.0
      </coordinates>
    </LinearRing>
  </innerBoundaryIs>
</Polygon>

```

```

        </coordinates>
    </LinearRing>
</innerBoundaryIs>
<innerBoundaryIs>
    <LinearRing>
        <coordinates>
            60.0,60.0 60.0,90.0 90.0,90.0 90.0,60.0 60.0,60.0
        </coordinates>
    </LinearRing>
</innerBoundaryIs>
</Polygon>

```

Para além destas classes, o GML define também colecções das mesmas. A título de exemplo, segue-se uma codificação de um *MultiLineString*:

```

<MultiLineString srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
    <lineStringMember>
        <LineString>
            <coord><X>56.1</X><Y>0.45</Y></coord>
            <coord><X>67.23</X><Y>0.98</Y></coord>
        </LineString>
    </lineStringMember>
    <lineStringMember>
        <LineString>
            <coord><X>46.71</X><Y>9.25</Y></coord>
            <coord><X>56.88</X><Y>10.44</Y></coord>
        </LineString>
    </lineStringMember>
    <lineStringMember>
        <LineString>
            <coord><X>324.1</X><Y>219.7</Y></coord>
            <coord><X>0.45</X><Y>4.56</Y></coord>
        </LineString>
    </lineStringMember>
</MultiLineString>

```

As outras classes de colecções são muito semelhantes.

Codificação de Objectos Geográficos

O GML fornece um conjunto de propriedades geométricas pré-definidas que podem ser usadas para relacionar propriedades geométricas de determinados tipos a objectos geográficos. O exemplo seguinte usa a propriedade *location*, que é um dos nomes descritivos pré-definidos:

```

<Dean>
    <familyName>Smith</familyName>
    <age>42</age>
    <nickName>Smithy</nickName>
    <nickName>Bonehead</nickName>
    <gml:location>
        <gml:Point>
            <gml:coord>

```

```
        <gml:X>1.0</gml:X><gml:Y>1.0</gml:Y>
      </gml:coord>
    </gml:Point>
  <gml:location>
</Dean>
```

Este exemplo serve também para mostrar o uso de esquemas aplicativos. Os elementos `<Dean>`, `<familyName>`, `<age>`, etc, não fazem parte da especificação GML. Estes elementos fazem parte do esquema definido para esta aplicação em particular. O esquema XML para o documento anterior é o seguinte:

```
<element name="Dean" type="ex:DeanType" substitutionGroup="gml:_Feature"/>

<complexType name="DeanType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <element name="familyName" type="string"/>
        <element name="age" type="integer"/>
        <element name="nickName" type="string" minOccurs="0"
          maxOccurs="unbounded"/>
        <element ref="gml:location"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Este esquema mostra como os elementos GML são estendidos de forma a dar-lhes um significado próprio no contexto da aplicação concreta.

Apêndice C

SVG

O SVG (*Scalable Vector Graphics*) é uma linguagem para descrever gráficos bi-dimensionais em XML. O SVG permite três tipos de objectos gráficos: formas gráficas vectoriais (e.g., polígonos), imagens e texto.

Os desenhos SVG podem ser interactivos e dinâmicos. Através de *scripting*, é possível manipular o DOM (*Document Object Model*) do SVG e ter acesso a todos os elementos, atributos e propriedades. Além disso, a norma define um conjunto de *event handlers* (e.g., `onmouseover` e `onclick`), que podem ser atribuídos a qualquer objecto gráfico do SVG.

A especificação SVG 1.1 (a versão mais recente na altura da escrita deste relatório) está dividida em módulos que fornecem unidades de funcionalidades específicas. Deste modo, os módulos podem ser combinados de forma a constituírem subconjuntos do SVG.

A modularização do SVG permite a definição de perfis, compostos por um determinado conjunto de módulos e possivelmente por um conjunto de restrições, ou extensões, aos elementos desses módulos.

Existem, neste momento, três perfis de SVG: o perfil *SVG Full*, e os perfis móveis: *SVG Tiny* e *SVG Basic*. O perfil *Full* inclui todos os módulos da especificação SVG. Quando usamos apenas o termo SVG estamos a referir-mo-nos ao perfil *SVG Full*. Os perfis *Tiny* e *Basic* foram definidos tendo como alvo pequenos dispositivos móveis, com limitações de recursos. O perfil *Tiny* é orientado para dispositivos muito limitados enquanto que o *Basic* é orientado para dispositivos de mais alto nível. O perfil *Tiny* é um subconjunto do perfil *Basic*, sendo este um subconjunto do SVG 1.1.

C.1 *Rendering*

O SVG usa o “modelo dos pintores”. Basicamente, significa que áreas sobrepostas são “pintadas por cima”, de acordo com os valores de transparência definidos. A sequência de desenho é a mesma dos objectos no ficheiro. Os objectos definidos primeiro, são pintados primeiro.

C.2 Estrutura do Documento SVG

Um documento SVG é composto por um elemento `<svg>`, que pode conter: outros elementos `<svg>`, elementos recipientes¹, ou elementos gráficos. Em SVG é possível definirem-se objectos para serem usados mais tarde, no documento. O elemento `<defs>` é usado para isso. Os objectos definidos em `<defs>` não são desenhados quando o elemento é interpretado, apenas o são quando é encontrada, no corpo do documento, uma referência a um objecto lá definido. A referência é feita através de um “id” definido aquando da definição do objecto.

O Exemplo C.2.1 mostra um documento SVG muito simples.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC
"-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg viewBox="0 0 200 200" xmlns="http://www.w3.org/2000/svg">

<defs>
  <rect id="myrect" x="0" y="0" width="10" height="10"/>
</defs>

<g>
  <rect fill="none" stroke="red" x="0" y="0" width="200" height="200"/>
  <use x="100" y="100" xlink:href="#myrect"/>
</g>

</svg>
```

Exemplo C.2.1: Exemplo de um documento SVG

O SVG permite fazer agrupamentos de objectos relacionados usando o elemento `<g>`. Os elementos `<desc>` e `<title>` podem ser usados dentro de um grupo para transmitir informação sobre a estrutura e semântica do documento. O elemento `<g>` é também útil para mudar o sistema de coordenadas dos objectos que contém. É possível atribuir uma transformação ao grupo de forma a que todos os objectos que estão dentro do grupo herdaram a transformação definida.

C.3 Caminhos (*Paths*)

Os caminhos representam a fronteira de formas que podem ser preenchidas, *stroked*, usadas como caminhos de recorte (*clipping*), ou uma combinação das três. Este elemento tem uma sintaxe muito poderosa que usa o conceito de ponto actual. A partir do ponto actual é possível desenhar-se uma recta ou curva até determinado ponto final, que passa a tornar-se o ponto actual. Também é possível mover-se o ponto actual sem se desenhar nada. Este conceito é usado noutros formatos gráficos como o *Postscript*. Os pontos do caminho podem ser especificados em coordenadas absolutas ou relativas.

O elemento `<path>` é usado para definir caminhos. O atributo “d”, *path data*, contém a definição do conteúdo do caminho.

A sintaxe de um caminho consiste no seguinte: uma letra é usada para definir a instrução. A

¹Um elemento recipiente é um elemento que pode conter elementos gráficos ou outros elementos recipientes.

lista instruções é dada na Tabela C.3.1. Uma instrução tem zero ou mais parâmetros que vêm a seguir à instrução. Todas as instruções têm uma forma relativa, isto é, uma forma em que os parâmetros são pontos relativos ao ponto actual. Uma letra maiúscula indica uma instrução absoluta, enquanto que uma letra minúscula indica uma instrução relativa.

Tabela C.3.1 Instruções do elemento `<path>`

Instrução	Nome	Parâmetros	Descrição
M/m	<i>moveto</i>	x y	Move o ponto actual para o ponto especificado.
Z/z	<i>closepath</i>	(nenhum)	“Fecha” o caminho.
L/l	<i>lineto</i>	x y	Desenha uma linha até ao ponto especificado.
H/h	<i>horizontal lineto</i>	x	Desenha uma linha horizontal.
V/v	<i>vertical lineto</i>	y	Desenha uma linha vertical.
C/c	<i>curveto</i>	x1 y1 x2 y2 x y	Desenha uma curva de Bézier desde o ponto actual até ao ponto (x y) usando (x1 y1) e (x2 y2) como pontos de controlo.
S/s	<i>smooth curveto</i>	x2 y2 x y	Desenha uma curva de Bézier desde o ponto actual até (x y). O primeiro ponto de controlo é a reflexão do segundo ponto de controlo da instrução anterior, relativa ao ponto actual.
Q/q	<i>quadratic Bézier curveto</i>	x1 y1 x y	Desenha uma curva de Bézier quadrática desde o ponto actual até ao ponto (x y) usando (x1 y1) como ponto de control.
T/t	<i>quadratic Bézier smooth curveto</i>	x y	Desenha uma curva de Bézier quadrática desde o ponto actual até (x y). O ponto de controlo é a reflexão do ponto de controlo da instrução anterior, relativa ao ponto actual.
A/a	<i>elliptical arc</i>	rx ry x-axis-rotation large-arc-flag sweep-flag x y	Desenha um arco desde o ponto actual até (x y).

O Exemplo C.3.1 define um caminho que representa um quadrado com 10 unidades de lado, com o vértice superior esquerdo no ponto (0, 0).

```
<path d="M 0 0 1 10 0 1 0 10 1 -10 0 1 0 -10z">
```

Exemplo C.3.1: Exemplo de um caminho

C.4 Elementos Básicos

O elemento `<path>` permite-nos desenhar qualquer forma possível em SVG. No entanto, existem alguns elementos que facilitam a definição de algumas formas muito usadas.

O SVG fornece as seguintes formas básicas:

- rectângulos
- círculos
- elipses
- linhas
- poli-linhas
- polígonos

O elemento `<rect>` permite desenhar rectângulos. É possível desenhar rectângulos com cantos redondos, atribuindo valores aos atributos “`rx`” e “`ry`”. Os outros atributos de `<rect>` são: “`x`” — a coordenada x do lado do rectângulo com menor coordenada x, “`y`” — a coordenada y do lado do rectângulo com menor coordenada y, “`width`” — a largura do rectângulo e “`height`” — a altura do rectângulo.

O elemento `<circle>` desenha um círculo, dados um ponto central e o raio. Os atributos de `<circle>` são: “`cx`” — a coordenada x do centro do círculo, “`cy`” — a coordenada y do centro do círculo e “`r`” — o raio do círculo.

Para desenhar elipses, é usado o elemento `<ellipse>`. Os atributos deste elemento são: “`cx`” — a coordenada x do centro da elipse, “`cy`” — a coordenada y do centro da elipse, “`rx`” — o raio na direcção do eixo dos xx e “`ry`” — o raio na direcção do eixo dos yy.

O elemento `<line>` desenha um segmento de recta, que começa num ponto e acaba noutra. Tem quatro atributos: “`x1`”, “`y1`”, “`x2`” e “`y2`” que definem as coordenadas dos dois pontos do segmento de recta.

Quando queremos desenhar um conjunto de linhas conectadas usamos o elemento `<polyline>`. Tem apenas um atributo: “`points`” — a lista de pontos que compõem a poli-linha. O número de coordenadas deve ser um número par, caso contrário o elemento está em erro. Tipicamente, este elemento define formas abertas.

O elemento `<polygon>` é usado para desenhar polígonos. Este elemento tem também um atributo “`points`” que define a lista de pontos que compõe o polígono. A diferença para o elemento `<polyline>` é que, neste caso, a forma é sempre desenhada fechada.

O Exemplo C.4.1 mostra alguns exemplos das formas básicas descritas. A Figura C.4.1 mostra o resultado gráfico do exemplo.

C.5 Transformações

É possível alterar o sistema de coordenadas actual especificando o atributo “`transform`”, num elemento recipiente ou num elemento gráfico. O atributo “`transform`” pode tomar várias formas. É possível especificar uma matriz de transformação ou especificar transformações simples do género rotação, translação, etc. É possível ainda definir uma sequência de transformações a serem aplicadas na ordem especificada. As transformações aceites pelo atributo são:

```

<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg viewBox="0 0 200 250" xmlns="http://www.w3.org/2000/svg">
<g>
  <desc>Dois retangulos: um com cantos retos, outro com cantos
    arredondados
  </desc>
  <rect fill="none" stroke="red" x="10" y="10" width="50" height="50"/>
  <rect fill="none" stroke="blue" x="70" y="10" rx="5" ry="5"
    width="50" height="50"/>
</g>
<g>
  <desc>Um circulo</desc>
  <circle r="25" cx="155" cy="35" fill="none" stroke="green"/>
</g>
<g>
  <desc>Uma elipse</desc>
  <ellipse rx="25" ry="12" cx="215" cy="35" fill="none"
    stroke="black"/>
</g>
<g>
  <desc>Uma linha e uma poli-linha</desc>
  <line x1="10" y1="70" x2="60" y2="120" stroke="black"/>
  <polyline points="70,70 70,120 90,70 90,120 110,70 110,120"
    stroke="black" fill="none"/>
</g>
<g>
  <desc>Um poligono</desc>
  <polygon points="130,70 130,120 155,100 180,120 180,70"
    stroke="black" fill="red"/>
</g>
</svg>

```

Exemplo C.4.1: Exemplo de formas básicas em SVG

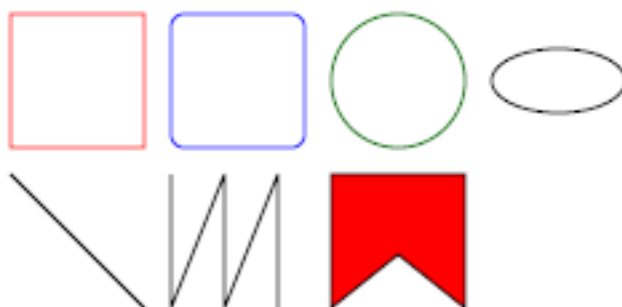


Figura C.4.1: Aspecto gráfico do Exemplo C.4.1

matrix($\langle a \rangle \langle b \rangle \langle c \rangle \langle d \rangle \langle e \rangle \langle f \rangle$) Isto é equivalente a definir a matriz de transformação 3x3:

$$\begin{pmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{pmatrix}$$

translate($\langle tx \rangle$ [$\langle ty \rangle$]) Especifica uma translação em x e em y. Se $\langle ty \rangle$ não for especificado, assume-se que é zero.

scale($\langle sx \rangle$ [$\langle sy \rangle$]) Especifica um aumento (ou redução) por um factor de sx e sy . Se $\langle sy \rangle$ não for especificado, assume-se que é igual a $\langle sx \rangle$.

rotate($\langle angulo \rangle$ [$\langle cx \rangle \langle cy \rangle$]) Especifica uma rotação de $angulo$ graus sobre um dado ponto. Se $\langle cx \rangle$ e $\langle cy \rangle$ não forem fornecidos, o ponto de rotação é a origem do sistema de coordenadas actual.

skewX($\langle angulo \rangle$) Especifica uma distorção sobre o eixo dos xx.

skewY($\langle angulo \rangle$) Especifica uma distorção sobre o eixo dos yy.

C.6 SVG Tiny

O SVG *Tiny* é um dos perfis móveis da norma SVG. Este perfil é direccionado a dispositivos móveis com sérias restrições. Uma vez que os dispositivos alvo têm muitas limitações de memória, de processamento e de visualização, o SVG *Tiny* introduz restrições ao conteúdo, propriedades e tipos de atributos do SVG.

As principais restrições introduzidas pelo perfil SVG *Tiny* são as seguintes:

Números O SVG *Tiny* suporta apenas números de vírgula fixa, na gama $-32,767.9999$ a $+32,767.9999$.

Cores Apenas suporta cores sólidas. Gradientes e padrões não são suportados.

Estrutura O SVG *Tiny* não suporta elementos $\langle svg \rangle$ aninhados.

Porcentagem SVG *Tiny* não suporta percentagens excepto nos atributos “width” e “height” do elemento $\langle svg \rangle$.

CSS Estilos com CSS não são suportados.

Caminhos As instruções *A* e *a* (*elliptical arc curve*) não são suportadas.

Transparência O SVG *Tiny* não suporta os atributos de transparência.

Scripting O SVG *Tiny* não suporta *scripting*.