Universidade do Porto
Faculdade de Engenharia
FEUP

Alberto Manuel Torres de Carvalho

Sistemas de Informação Geográfica em Dispositivos Móveis

Gestão e Visualização de Cartografia Militar

Junho, 2005

Faculdade de Engenharia da Universidade do Porto

Licenciatura em Engenharia Informática e Computação

Sistemas de Informação Geográfica em Dispositivos Móveis na ParadigmaXis, S.A.

Relatório do Estágio Curricular da LEIC 2004/2005

Alberto Manuel Torres de Carvalho

Orientador na FEUP: Prof. António Augusto Sousa

Orientador na ParadigmaXis, S.A.: Prof. Alexandre Sousa

Junho de 2005

Resumo

Este documento contém o relatório de estágio realizado na empresa *ParadigmaXis, S.A.*, relativo à disciplina "Estágio Curricular" da Licenciatura em Engenharia em Informática e Computação da FEUP. O projecto está subordinado ao tema "Sistemas de Informação Geográfica em Dispositivos Móveis", com o sub-tema "Gestão e Visualização de Cartografia Militar".

O termo *Sistemas de Informação Geográfica* (SIG) aplica-se a sistemas que realizam o tratamento computacional de dados geográficos e recuperam informações, não apenas com base nas suas características alfanuméricas, mas também através da sua localização espacial [Câmara, 1996]. Os SIG são ferramentas indispensáveis na sociedade tecnológica em que vivemos. Actualmente, a sua ampla gama de aplicações inclui áreas como a agricultura, cartografia, gestão de redes de comunicação e energia, sistemas de navegação, gestão de recursos, entre outras.

Nos últimos anos, a evolução dos SIG tem sofrido avanços tecnológicos significativos. Os factores impulsionadores desta evolução têm sido o notável aumento da capacidade de processamento conjuntamente com a elevada capacidade de armazenamento computacional de que actualmente dispomos. Concomitantemente com esta evolução, desenvolveram-se infraestruturas tecnológicas modernas e assistimos à massificação da utilização da Internet e das tecnologias móveis [Jonh et al., 2003]. A conjugação destes factores criou o ambiente ideal para o surgimento de aplicações SIG no domínio da Internet e das tecnologias móveis. É nesta última área que se enquadra o trabalho descrito neste documento.

O projecto tem como objectivo desenvolver um sistema de visualização de informação georreferenciada para dispositivos móveis do tipo *Pocket PC*. A informação a ser visualizada inclui mapas, cartografia militar em formato *raster* e cartografia vectorial. O sistema disponibiliza algumas das funcionalidades típicas de um SIG, tais como operações comuns de navegação sobre um mapa (*zoom in/out/all* e *pan down/up/left/righ*) e operações de pesquisa alfanumérica com localização.

O trabalho desenvolvido é constituído por duas aplicações complementares – *MapAdventure* e *MapAdventure Studio*. O *MapAdventure Studio* é uma aplicação para a plataforma *Windows Desktop* que permite exportar informação geográfica – cartografia militar em formato *raster* – para o dispositivo móvel. O *MapAdventure* é uma aplicação para dispositivos móveis *Pocket PC* que disponibiliza ao utilizador a visualização de cartografia *raster* (importada do *MapAdventure Studio*) e vectorial. Além das funcionalidades típicas de uma aplicação SIG, o *MapAdventure* contém um módulo de interligação com um dispositivo *GPS*[1] de forma a permitir a localização, em tempo real, do utilizador.

O projecto encontra-se, no momento de escrita deste documento, incompleto, pois a aplicação *MapAdventure* ainda se encontra na fase de implementação. Contudo, a aplicação *MapAdventure Studio* encontra-se prestes a entrar na fase de comercialização por parte da *ParadigmaXis*.

[1] Do inglês *Global Positioning System.*

Agradecimentos

Gostaria da agradecer ao meu orientador de estágio por parte da *ParadigmaXis, S.A.*, Prof. Alexandre Valente Sousa por me ter proporcionado esta oportunidade de estágio e pela sua disponibilidade durante a realização deste projecto. De igual modo, agradeço ao orientador de estágio por parte da FEUP, Prof. António Augusto de Sousa, pelo enorme apoio e disponibilidade durante o desenrolar do estágio. O meu agradecimento vai também para todos os colegas da *ParadigmaXis* que tanto contribuíram para a integração na empresa e para um ambiente de trabalho agradável. Um agradecimento especial para o Eng. José Moreira pelo enorme apoio arquitectónico e tecnológico ao longo de todo o projecto, que tanto contribuíu para o sucesso deste projecto.

Os meus agradecimentos vão também para toda a minha família e amigos que me apoiaram, não só durante a realização do estágio, mas também ao longo de todo o curso.

Um agradecimento especial para a Telma pela enorme paciência e apoio ao longo de todo o curso.

Glossário

API Abreviatura de *Application Program Interface*. Consiste num conjunto de rotinas, protocolos e ferramentas que permitem produzir aplicações de *software*.

Assembly O conjunto de um ou mais ficheiros identificado por uma versão e instalado como sendo uma unidade. Uma *assembly* é o bloco fundamental de construção de uma aplicação para a *.NET Framework* e para a *.NET Compact Framework*. Todos os tipos de dados e recursos necessários são contidos numa *assembly* e marcados com modificadores de visibilidade. O *assembly* é constituído por meta-informação e *MSIL*.

C# Linguagem de programação orientada aos objectos, desenvolvida pela *Microsoft*, largamente similar ao *C++* e ao *JAVA*.

Carta Militar Representação gráfica da superfície da terra ou de parte dela. É desenhada num determinado plano e escala.

Cartografia *Raster* Representação, através de imagens georreferenciadas, da topografia de uma determinada área da superfície terrestre.

Cartografia Vectorial Representação, através de informação vectorial, da topografia de uma determinada área da superficie terrestre.

CLR abreviatura de *Common Language Runtime*. Consiste num ambiente *runtime* que gere a execução de código *MSIL* e providencia, entre outros, serviços de gestão de memória, gestão de excepções, depuração, *profiling* e segurança. O *CLR* é um componente essêncial da *.NET Framework*.

Código nativo Também referido como código máquina refere-se ao código que é escrito para um processador específico usando o conjunto de instruções desse processador.

Emulador Programa ou dispositivo que imita outro programa ou dispositivo. No contexto deste documento refere-se ao programa que simula o funcionamento do dispositivo do tipo *Pocket PC* numa plataforma *Windows Desktop*.

Ferramentas de Depuração Ferramentas computacionais que ajudam o programador a encontrar e a resolver erros de programação.

Double Buffer Técnica bastante conhecida que permite optimizar o desempenho gráfico de uma aplicação através da criação de um *buffer* em memória, onde são efectuadas todas as operações a nível gráfico. Na fase de desenho, apenas se copia o *buffer* para o ecrã.

Dispositivo móvel Dispositivo que possui algumas das funcionalidades existentes no PC, embora com uma capacidade de processamento e armazenamento inferior. No contexto deste documento, refere-se ao dispositivo *Pocket PC*.

GPS Sistema que permite determinar a posição exacta de qualquer objecto na superficie terrestre através do uso de satélites e equações matemáticas.

Interface gráfica Termo que engloba todas características e elementos gráficos que um programa de computador apresenta para facilitar a interacção com o utilizador.

Interoperação No contexto deste documento, indica os vários modos de interacção entre *managed code* e *unmanaged code*. Existem dois tipos de interoperação: através do mecanismo *Platform Invoke* ou através da utilização objectos *COM*.

JIT Abreviatura de *Just-In-Time*. Quando usado no contexto de compilação (*JIT Compiler*), consiste numa estratégia em que o código é compilado durante a sua execução.

Managed Code Código que é compilado para *MSIL* e que, posteriormente é compilado pelo *CLR* para código-máquina.

Meta-Informação Termo que significa "informação sobre informação". Meta-informação são os dados sobre um dado documento (por oposição aos conteúdos do documento propriamente ditos).

MSIL *Microsoft Intermediate Language* consiste na linguagem intermédia para qual todas as linguagens de alto-nível da *.NET Framework* compilam, antes de serem convertidas para código máquina pelo compilador *JIT*.

Namespace No âmbito da tecnologia *.NET* corresponde a uma biblioteca de classes.

Objectos COM Arquitectura desenvolvida pela *Microsoft* para construir aplicações baseadas em componentes. Os objectos COM expõem interfaces que permitem a outras aplicações aceder às suas funcionalidades.

P/Invoke Mecanismo que permite aceder a *unmanaged code* a partir de um ambiente *managed*.

Pan Capacidade de um sistema de visualização mostrar vários segmentos da imagem sem mudar a escala de visualização. No contexto deste documento, refere-se à capacidade de o utilizador "mover" o mapa.

Rota Conjunto de pontos que definem um percurso desde a origem até ao destino.

SIG Abreviatura de Sistema de Informação Geográfica.

Software Instruções computáveis por determinado *hardware* com a finalidade de levar a cabo uma determinada tarefa.

Topografia Representação da configuração de um terreno ou de uma localidade, com todos os seus "acidentes" geográficos;

Track No contexto deste documento, refere-se ao registo de um percurso, percorrido por alguém, através de vários pontos intermédios.

Thread Parte de um programa que pode ser executada de um modo independente relativamente a outras partes do programa. Permite a paralelização de vários métodos dentro de um mesmo programa.

Umanaged Code Código que é desenvolvido sem as convenções e limitações da plataforma *.NET* . Principalmente utilizado para: efectuar chamadas a funções da *API* do sistema operativo; desenvolver objectos *COM*; aceder a áreas críticas de memória; e desenvolver rotinas de desempenho crítico). Este código é executado fora do âmbito do *CLR*.

Waypoint Ponto intermédio de uma *track* ou de uma rota.

Zoom Função que permite variar as áreas de visualização de uma imagem. Existem, principalmente, dois tipos de *zoom*: O *zoom in* (ampliação) representa uma área seleccionada da imagem numa escala mais reduzida; o *zoom out* (redução) representa uma área seleccionada da imagem numa escala maior.

Zoom to fit No contexto deste documento significa a adaptação do nível de *zoom* do mapa à área a visualizar.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Este capítulo introduz o presente documento, bem como a empresa onde decorreu o estágio curricular. O capítulo contempla ainda o contexto em que se realiza o projecto, os objectivos definidos para o projecto e, no final, a organização deste documento, sendo uma breve descrição de cada capítulo. No fim, o leitor deve possuir uma visão global sobre o que será tratado ao longo do documento.

1.1 A empresa – *ParadigmaXis, S.A.*

A *ParadigmaXis — Arquitectura e Engenharia de Software S.A.* é uma empresa nacional, criada em 2000, especializada na construção e integração de sistemas de *software* dedicados ao processamento de informação espaço-temporal.

A *ParadigmaXis* cria soluções à medida, recorrendo a conceituadas práticas de engenharia de *software* e reutilizando componentes normalizados. Para o desenvolvimento de *software* utiliza a prática conjunta de *eXtreme Programming* [Beck, 2000], reutilização de *software* [Coulange, 1998], padrões de *software* [Gamma et al., 1995], paradigma de orientação por objectos [Graham, 2001] e programação literária [Knuth, 1984].

A *ParadigmaXis* resultou da iniciativa de um grupo de pessoas com uma vasta experiência no domínio da Engenharia de *Software*. Desde o seu início que tem privilegiado uma actuação em rede, em complementaridade com outros parceiros empresariais, que de alguma forma tragam valor acrescentado às actividades por ela desenvolvida.

É nesta direcção que, em Maio de 2001 a *ParadigmaXis* realizou um *spin-off*, através da parceria com a *Procme — Gestão Global de Projectos S.A.*, originando a *Ubicuos — Sistemas de Georeferenciação S.A.*. A *ParadigmaXis* é também membro da *Oracle Partner Program* e do *Autodesk Developer Network.*

1.2 Sistemas de Informação Geográfica

Um Sistema de Informação Geográfica é constituído por um conjunto de "ferramentas" especializadas em adquirir, armazenar, recuperar, transformar e emitir informações espaciais.

Manipula assim dados que geográficos descrevem objectos do mundo real em termos de posicionamento (relativamente a um sistema de coordenadas), atributos (como a cor, pH, custo, incidência de pragas, etc) e das relações topológicas existentes. Sendo assim, um SIG pode ser utilizado em estudos relativos ao meio ambiente e recursos naturais, na pesquisa da previsão de determinados fenómenos ou no apoio a decisões de planeamento; [Burrough, 1986].

Estes sistemas oferecem ao utilizador uma visão inédita do seu ambiente de trabalho, em que todas as informações disponíveis sobre um determinado assunto estão ao seu alcance, interrelacionadas com base no que lhes é fundamentalmente comum - a localização geográfica [Chang, 2002]. Para que tal seja possível, a geometria e os atributos dos dados num SIG devem estar georreferenciados, isto é, localizados na superfície terrestre e representados numa projecção cartográfica.

O requisito de armazenar informação relativa à geometria dos objectos geográficos e dos seus atributos representa uma *dualidade* básica para os SIG [Heywood et al., 1998]. A importância destes sistemas reside, exactamente, nesta capacidade de armazenar informação de diversos domínios e simultaneamente processar a sua integração num ambiente georreferenciado. Assim, para cada objecto geográfico, o SIG necessita armazenar os seus atributos e as várias representações gráficas associadas.

Existem pelo menos três grandes modos de utilizar um SIG:

- como ferramenta para produção de mapas;
- como suporte para análise espacial de fenómenos;
- como uma base de dados geográficos, com funções de armazenamento e recuperação de informação espacial.

Actualmente, assistimos a uma intensa proliferação dos SIG na nossa sociedade. Esta expansão deve-se principalmente aos avanços tecnológicos na área do poder computacional.

A evolução tecnológica possibilitou:

- o uso de base de dados mais refinadas;
- avanços nos processos de análise, modelação e visualização de dados – *hardware* e *software;*
- o desenvolvimento de interfaces gráficas "*user-friendly*"[2] que facilitaram o uso de aplicações computacionais complexas;
- o aumento de funcionalidades disponibilizadas ao utilizador.

Todos estes factores combinados com a rápida difusão dos computadores pessoais e, desde 1992, com o rápido crescimento do comércio na Internet, contribuiram para a rápida difusão dos SIG [Demers, 1997]. Tal como outros sistemas de informação, os SIG beneficiaram notavelmente da expansão da Internet: hoje em dia, aplicações SIG estão disponíveis na Internet através de "*web services*" e, em alguns casos, um único servidor responde a milhões de pedidos por semana.

[2] Termo que designa a fácil usabilidade de um dado sistema.

No domínio das tecnologias móveis verifica-se o "*boom*" das aplicações baseadas na localização[3], onde aplicações SIG estão instaladas em pequenos dispositivos móveis. Nesta área podem-se considerar principalmente dois tipos de aplicações: as que trocam informação com servidores dedicados, de modo a fornecer informação detalhada ao utilizador, e as que integram versões simplificadas de aplicações SIG comerciais no dispositivo [Pick, 2004]. Actualmente, este tipo de aplicações é constituído por um módulo *GPS* que permite efectuar a localização do utilizador em tempo real. A combinação do *GPS* com os SIG permite que o utilizador disponha de toda uma nova gama de funcionalidades. Entre estas, destacam-se os serviços baseados na localização. Através deste tipo de funcionalidades, o utilizador pode, por exemplo, saber qual a estrada a seguir para chegar mais rapidamente ao seu destino, qual o melhor restaurante na zona onde se encontra e até saber o menu disponível no restaurante para um determinado dia.

1.3 Objectivos

O objectivo deste estágio prende-se com o desenvolvimento de um sistema de informação geográfica para dispositivos móveis *Pocket PC*, utilizando a tecnologia e o *know-how* desenvolvidos na *ParadigmaXis*. A informação a ser visualizada inclui cartografia vectorial e cartografia militar em formato *raster*.

A limitação imposta pela reduzida capacidade de armazenamento e processamento de um dispositivo móvel representa uma variável incontornável para este projecto e condiciona todas as fases de desenvolvimento da aplicação. Por outro lado, no ínicio do projecto, foi definido um importante requisito referente à usabilidade. Neste domínio, toda a interface gráfica deve ser desenhada de modo a permitir uma rápida curva de aprendizagem por parte de qualquer tipo de utilizador. Com base nestas premissas foram especificadas duas aplicações que definem os principais objectivos do projecto.

O primeiro objectivo contempla o desenvolvimento da aplicação *MapAdventure Studio* para a plataforma *Windows Desktop*. A principal função desta aplicação é manipular e formatar a informação recebida do Instituto Geográfico do Exército (IGEOE) para poder ser utilizada pela aplicação *MapAdventure*. Esta fase inclui uma outra tarefa que visa implementar o desenvolvimento da interface de comunicação com a aplicação *MapAdventure*.

O segundo objectivo corresponde ao desenvolvimento da aplicação *MapAdventure* para a plataforma *Pocket PC*. Esta aplicação utiliza a informação fornecida pela aplicação *MapAventure Studio* para disponibilizar ao utilizador cartografia *raster* e vectorial. Adicionalmente, a aplicação integra um módulo de interligação com o GPS que permite actualizar a localização do utilizador em tempo real e possibilita a utilização das funcionalidades de *tracking* que efectuam o registo geográfico do caminho percorrido pelo utilizador. Esta aplicação é desenvolvida como prova de conceito, de modo a testar as capacidades da nova plataforma da *Microsoft* para dispositivos móveis, denominada *.NET Compact Framework* (*.NETCF*[4]). Mais concretamente, pretende-se testar as capacidades da plataforma em termos de desempenho gráfico.

---

[3] Do inglês "*location-based applications*".

[4] No decorrer do texto será utilizada a sigla *.NETCF* como abreviatura de *.NET Compact Framework*.

1.4 Organização do Presente Relatório

O presente capítulo é a introdução ao documento e apresenta o projecto de estágio, assim como a empresa onde o estágio foi realizado.

O segundo capítulo destina-se a apresentar de uma forma mais detalhada todo o contexto em que o problema se insere e identificar os vários problemas que o seu desenvolvimento implica.

O terceiro capítulo é constituido por duas partes. Na primeira, são apresentadas algumas soluções semelhantes para o problema em questão; são especificadas as características de cada solução e é feita uma breve descrição dos prós e contras de cada solução. A segunda parte descreve o conjunto de tecnologias envolvidas na realização deste projecto; neste domínio, são apresentadas as principais características da *.NET Framework* e *.NET Compact Framework* e é feita uma apresentação da plataforma de desenvolvimento *eMbedded Visual C++*.

O quarto capítulo apresenta requisitos funcionais e não funcionais definidos para o sistema a implementar, bem como a descrição da respectiva arquitectura lógica do sistema a desenvolver. Numa primeira fase é apresentada a arquitectura comum às duas aplicações e, posteriormente, são descritas as particularidades arquitecturais de cada aplicação.

O quinto capítulo descreve os pormenores da implementação do sistema desenvolvido, nomeadamente ao nível das classes mais relevantes das duas aplicações. Tal como na capítulo anterior, existe uma secção dedicada à implementação comum às duas aplicações e, posteriormente, são apresentadas secções que descrevem as classes exclusivas a cada aplicação.

O último capítulo apresenta as conclusões tiradas no final do projecto, nomeadamente os objectivos cumpridos e conhecimentos adquiridos, assim como algum trabalho futuro a ser efectuado.

Capítulo 2

Análise do problema

Este capítulo destina-se a efectuar uma apresentação do problema. A primeira parte descreve o contexto em que o projecto se enquadra. Após uma descrição sumária dos objectivos do projecto, segue-se uma apresentação dos requisitos gerais inicialmente propostos para o sistema, assim como soluções para algumas das questões com ele relacionadas.

2.1 Introdução

A *ParadigmaXis* tem vindo a desenvolver tecnologia na área do processamento e manipulação de informação georreferenciada. Esta tecnologia permite o desenvolvimento de motores gráficos que possibilitam a geração de mapas interactivos usados pela *ParadigmaXis*. Nesta área, foram já desenvolvidas várias aplicações que permitem disponibilizar, em plataformas móveis, serviços que recorrem a uma base de informação espacial e alfanumérica que se encontra parcialmente residente no dispositivo móvel.

Neste caso particular, pretende-se desenvolver um sistema de visualização de cartografia *raster* e vectorial para dispositivos móveis *Pocket PC*. O sistema a desenvolver permite disponibilizar funcionalidades de navegação *off-road*[5] para o utilizador comum. Este produto destina-se principalmente à utilização nos desportos aventura e na navegação ao ar livre, não dispondo por isso de funcionalidades utilizadas nos sistemas de navegação rodoviária como, por exemplo, o cálculo automático de rotas.

Apesar do *know-how* existente na empresa relativamente a este tipo de sistemas, este projecto apresenta-se como um interessante desafio. O projecto revela-se bastante apelativo não só pelas áreas tecnológicas envolvidas, mas também pelo desafio inerente ao desenho e implementação do projecto. Neste campo há que considerar algumas questões importantes, a primeira das quais desde logo relacionada com as limitações impostas pelo dispositivo móvel ao sistema a desenvolver, nomeadamente a sua capacidade de processamento e armazenamento limitada.

[5] Navegação que não é efectuada em ambientes urbanos. Neste contexto, refere-se à navegação realizada em vias secundárias, ao ar livre ou em locais remotos.

Estes factores são importantes, pois influenciam as várias fases do desenvolvimento do projecto, desde a fase de desenho até à fase de implementação e, se não forem tomados em conta, podem por em causa o sucesso de todo o projecto. Assim sendo, torna-se necessário reflectir em algumas questões relativas ao contexto deste projecto:

- Como projectar a arquitectura do sistema de modo a contribuir para a eficiência da aplicação?
- Qual o formato dos dados da cartografia vectorial de forma a ocupar pouco espaço no dispositivo ?
- Quais as optimizações que se podem efectuar, a nível da implementação, de modo tornar a aplicação mais rápida com uma ocupação mínima de memória pelo código?
- Como projectar e implementar a interface gráfica de modo a ser apelativa, ocupar pouco espaço em memória com um *loading time*[6] mínimo ?
- Como projectar a interface gráfica de modo a obter uma curva de aprendizagem rápida?

Estas questões revestem-se de uma importância fundamental quando se aplicam a uma plataforma - *.NETCF* - que é utilizada pela primeira vez pela empresa para o desenvolvimento de aplicações para dispositivos móveis. Assim sendo, torna-se imprescindível obter as respostas certas a estas questões, para assegurar o sucesso do projecto. Para tal, é fundamental analisar as tecnologias utilizadas no desenvolvimento do projecto (ver secção 3.2) de modo a identificar os seus pontos fortes e as suas vulnerabilidades.

A última questão é bastante pertinente, pois no contexto deste tipo de aplicações, a curva de aprendizagem é um factor importante já que se trata de uma aplicação que é considerada novidade e que é destinada ao mercado dos dispositivos móveis. Portanto este factor é relevante para o desenrolar do projecto e não pode ser negligenciado.

No entanto, um bom projecto não significa um projecto de sucesso. Isto é, o sistema não pode ser apenas cuidadosamente projectado e desenvolvido, mas terá de possuir as características necessárias para ser útil e atraente para o mercado-alvo a que se destina. Além disso, necessita de oferecer ao utilizador algo novo e simultaneamente útil, de modo a que tenha sucesso num mercado tão competitivo como é o dos sistemas de navegação para plataformas móveis. Torna-se assim indispensável uma análise crítica das soluções existentes nesta área (ver secção 3.1) de forma a retirar algumas conclusões acerca da parcela de inovação do sistema a implementar. Deste modo será possível projectar um sistema competitivo e inovador que vença no mercado das plataformas móveis.

Este projecto desenrola um papel fundamental para a empresa, pois pretende agir como prova de conceito de modo a testar as capacidades da *.NETCF*. Mais concretamente, pretende-se testar as capacidades gráficas da *.NETCF* de forma a permitir à empresa optar pelo uso desta *framework* para este e outros projectos, ou continuar a usar as ferramentas computacionais até agora utilizadas no desenvolvimento de aplicações para dispositivos móveis que envolvam capacidades gráficas elevadas.

---

[6] Tempo que a aplicação demora a inicializar.

2.2 Requisitos Gerais do Sistema

Numa fase inicial de análise do problema, foram identificados alguns problemas que devido à sua importância para o posterior desenvolvimento e planeamento do projecto foram resolvidos numa primeira abordagem. De seguida, descrevem-se as decisões tomadas nesta fase.

Como já foi referido anteriormente, a limitação imposta pela reduzida capacidade de armazenamento e processamento do dispositivo móvel representa uma condicionante incontornável para este projecto. Tendo em conta este facto, foi projectado um sistema que permite visualizar cartografia – *raster* e vectorial – de um modo eficiente no dispositivo móvel. No entanto, desde logo se tornou óbvio que a totalidade da cartografia *raster* não poderia residir no dispositivo móvel em virtude do pouco espaço disponível. De modo a contornar este problema foi idealizada uma aplicação complementar – *MapAdventure Studio*, para a plataforma *Windows Desktop*, que possibilita seleccionar e exportar para o dispositivo apenas a cartografia *raster* correspondente às regiões que o utilizador pretende visualizar.

A aplicação *MapAdventure* permite ao utilizador construir *tracks* através dos caminhos percorridos pelo utilizador. Posteriormente, o utilizador poderá percorrer o mesmo caminho através da *track* que registou no seu dispositivo móvel. Esta aplicação disponibiliza ainda informação acerca da posição (no mapa ou através de coordenadas), velocidade (instântanea e média) e altitude do utilizador. A aplicação disponibiliza ainda cronómetros e diversos modos de registar *tracks*. Por sua vez, a aplicação *MapAdventure Studio* permite ao utilizador navegar no mapa e exportar regiões do mapa (em formato *raster*) para o dispositivo móvel. Esta aplicações disponibiliza ainda operações de pesquisa sobre vias e localidades do mapa. A operação de exportação de uma determinada região só é permitida caso o utilizador tenha adquirido a carta militar correspondente. Caso contrário, a aplicação permite operações de navegação, selecção e pesquisa, mas não permite a exportação. O processo de aquisição de cartas militares baseia-se na compra de determinados número de cartas. Aquando do processo de exportação, a aplicação *MapAdventure Studio* pede ao utilizador que este localize as cartas e, após a localização e validação, a aplicação efectua as cartas a exportar.

Na fase inicial, foi ainda definido mais um importante requisito para o sistema: a arquitectura da aplicação *MapAdventure Studio* deve ser re-utilizável pela aplicação *MapAdventure*. Assim, a arquitectura da aplicação *MapAdventure Studio* tem de ser idealizada de forma a ser modular e suficientemente flexível para ser facilmente adaptada para a aplicação *MapAdventure*. Esta aproximação é vantajosa, pois reduz o tempo de implementação do todo o sistema e, simultaneamente, é testada a portabilidade e compatibilidade de código entre as plataformas *.NET Framework* e *.NET Compact Framework*. No entanto, poderá envolver algum risco se a arquitecura desenvolvida inicialmente não se adaptar facilmente à plataforma móvel. Espera-se que não existam incompatibilidades que influenciem significativamente o normal decorrer do projecto.

No mesmo sentido do requisito descrito no parágrafo anterior, rapidamente se concluiu que a arquitectura deveria ser projectada de modo a suportar posteriores adaptações e/ou evoluções em ambas as aplicações. Adicionalmente, foi ainda definido que o desenvolvimento das aplicações deveria ser efectuado usando a linguagem de programação *C#* [Marques and Pedroso, 2002] de modo a tirar partido de três características distintas das linguagens de programação orientadas aos objectos: abstracção de dados, polimorfismo e herança.

Após esta fase foi então delineado o planeamento do projecto que resultou em duas fases que correspondem ao desenvolvimento das duas aplicações. As duas fases principais serão

executadas sequencialmente, sendo que a primeira fase corresponde ao desenvolvimento da aplicação *MapAdventure Studio* para a palataforma *Windows Desktop* e a segunda ao desenvolvimento da aplicação *MapAdventure* para a plataforma *Pocket PC*.

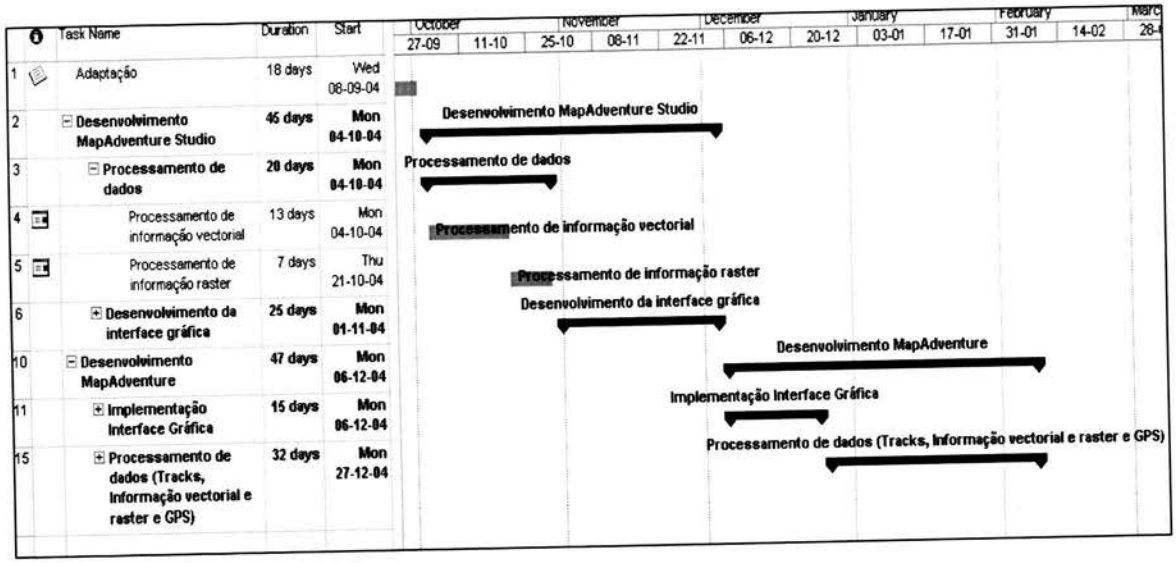

Figura 2.1 – Planeamento do projecto

Capítulo 3

Revisão Tecnológica

Este capítulo é constituido, essencialmente, por duas partes. Na primeira parte é apresentado o estado da arte na área circundante ao problema em causa; pretende-se apresentar ao leitor uma ideia clara acerca das eventuais soluções existentes para o problema ou para problemas semelhantes. Na segunda parte são descritas as tecnologias utilizadas ao longo do desenvolvimento do projecto, nomeadamente as plataformas de desenvolvimento da *Microsoft –.NET Framework* e *.NET Compact Framework*. Finalmente, é apresentado um conjunto de ferramentas adicionais utilizadas durante a realização deste projecto.

3.1 Soluções Existentes ou Semelhantes

Devido à proliferação de soluções no mercado que visam a manipulação de dados geográficos em plataformas móveis, foi feita uma síntese de modo a assegurar a diversidade de tecnologias apreentadas. A análise das várias soluções contempla a descrição da tecnologias usadas, funcionalidades disponibilizadas e a análise dos pontos fortes e dos pontos fracos relativamente a sistemas semelhantes.

3.1.1 *TomTom Navigator 3*

O *TomTom Navigator* 3 é o terceiro de uma série de sucesso de sistemas de navegação por satélite (*GPS*) destinado às plataformas *Pocket PC* e *Smartphone*. Este sistema destina-se principalmente à navegação rodoviária, visto que grande parte das funcionalidades são orientadas para o utilizador comum que se desloca de automóvel, embora também possa ser utilizado eficazmente para a navegação pedestre. Essencialmente, o sistema disponibiliza um mapa baseado em cartografia vectorial que actualiza, em tempo-real, a posicão do utilizador através do *GPS*. Contudo, o que torna a aplicação apelativa (mesmo para os utilizadores mais cépticos relativamente a estes sistemas) é o abrangente conjunto de funcionalidades que são oferecidas [PocketGPS, 2005]. De seguida são indicadas as funcionalidades de maior relevância disponibilizadas pelo sistema:

- Mapa com opção de visualização em 2D ou 3D e possibilidade de visualização de cores em modo nocturno ou diurno. O mapa disponibiliza funcionalidades típicas de navegação como o *zoom in/out* e *pan left/right/up/down*.
- Mapa constantemente sincronizado com a posição e direcção do utilizador.
- Planeamento de rotas com opção de inclusão de múltiplos pontos de passagem (*waypoints*) e possibilidade de replaneamento durante viagem. Ao longo da viagem existe a possibilidade de visualização do tempo restante até ao destino, tempo decorrido, distância percorrida e distância a percorrer.
- Informação de tráfego rodoviário através de uma ligação *GPRS*[7], permitindo deste modo evitar zonas congestionadas ou acidentadas em tempo-real.
- Cálculo de rotas para obtenção do caminho mais rápido ou mais curto.
- Alteração automática de rotas de modo a evitar acidentes, zonas de congestão de tráfego ou simplesmente evitar estradas que não queremos percorrer. O sistema calcula automaticamente uma rota alternativa no caso do utilizador se "desviar" da rota predefinida.
- Possibilidade de registo e exportação de rotas de modo a permitir a partilha com outros utilizadores.
- Extensa lista de pontos de interesse (hotéis, teatros, postos de abastecimento, etc.) indicados no mapa, através de ícones. A aplicação permite activar alarmes sonoros e gráficos que indicam a proximidade de pontos de interesse pré-seleccionados e permite criar os pontos de interesse pessoais.
- Permite efectuar procuras de pontos de interesse, localidades, ruas e endereços na base de dados interna.
- Indicação do destino da rota através de pontos de interesse, seleccção no mapa ou endereço de destino.
- Instruções de navegação através de pictogramas direccionais e indicações verbais (ex.: "Após 100 metros vire à direita").
- O módulo relativo ao *GPS* permite configurar a ligação ao dispositivo externo e possui várias indicações úteis: nível de sinal recebido, número de satélites e respectiva posição no espaço, direcção, velocidade, altitude e coordenadas (longitude e latitude) da posição do utilizador.

Prós:

- Instruções de navegação através de indicações verbais.
- Facilidade em encontrar o local pretendido.

[7] Abreviatura de *General Packet Radio Service*. Tecnologia que permite a transmissão de dados (*e-mail, acesso Web*) sobre uma rede *GSM*.

- Elevado número de opções, funcionalidades e *add-ons*[8].
- Interface gráfica intuitiva e fácil de usar.
- Instalação fácil e rápida.

Contras:

- Alguns mapas estão desactualizados.
- Elevada depêndencia da localização por satélite.
- Elevado *loading time*.

Embora esta aplicação se destaque pela diversidade de funcionalidades oferecidas, possui outras características que a tornam um sucesso entre os adeptos deste tipo de sistemas, de onde se destacam a interface intuitiva e o elevado nível de personalização das funcionalidades [Totalpda, 2005]. Contudo, a fácil instalação e o número reduzido de passos para aceder às funcionalidade são também factores que determinam o sucesso deste sistema. A combinação destas características aumenta o número de potenciais utilizadores, pois possibilita o uso do sistema por utilizadores mais experientes que pretendem adaptar o sistema às suas necessidades específicas e, simultaneamente, permite que utilizadores menos experientes possam utilizar a aplicação sem necessidade de efectuar operações complexas [Pocketpcmag, 2005].

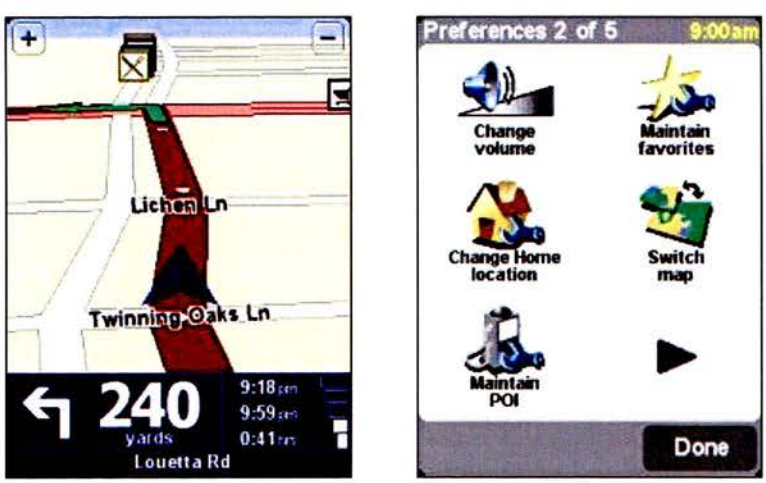

Figura 3.1 – *Screenshots* da aplicação *TomTom Navigator*

Além deste sistema, existem ainda outras soluções bastante referenciadas nesta área. Entre estas, destacam-se o *Destinator 3* e o *ALK CoPilot Live 5* que não foram contemplados nesta secção devido à similariedade com o sistema apresentado. Estas aplicações apresentam algumas diferenças ao nível da interface gráfica, mas de um modo geral, a tecnologia utilizada e as funcionalidades oferecidas são idênticas.

[8] Módulo de *software* adicional que permite aumentar a eficiciência e/ou funcionalidade do sistema ao qual é adicionado.

3.1.2 *Ozi Explorer CE*

Tal como o sistema anterior, disponibiliza no mapa a posição do utilizador em tempo real. No entanto, distingue-se dos sistemas de navegação rodoviária, sendo mais direccionado para a navegação *off-road*. Ao contrário do *TomTom Navigator 3*, o *Ozi Explorer CE* é fundamentalmente direccionado para desportos de aventura ou outras actividades (navegação marítima e/ou aérea) que envolvam a utilização de mapas (topográficos, aéreos, marítimos, etc.) [Pocketgps, 2005]. Existem duas diferenças cruciais entre estes dois sistemas: a primeira situa-se ao nível do formato dos mapas que neste caso são baseados em formato *raster* e não em cartografia vectorial. A segunda situa-se ao nível do cálculo automático de rotas que não existe no *OziExplorer CE*. Dado que, neste sistema, a prioridade é dada às funcionalidades que permitem a construção manual de rotas ou *tracks*. De seguida, destacam-se as funcionalidades mais importantes desta aplicação.

Mapa:

- Permite sincronizar o mapa com a posição do utilizador em tempo-real (se tiver o *GPS* activo).
- Disponibiliza funcionalidades típicas como o *zoom in/out* e *pan left/right/up/down*.
- Permite a criação e edição de diversos tipos de objectos (*waypoints*, *tracks*, rotas e comentários) no mapa.
- De forma a permitir a gestão de informação diversa no mapa a aplicação permite "mostrar" ou "esconder" os objectos do mapa e seleccionar a cor das várias *tracks* e rotas.

Rotas:

- Criação manual de rotas com inclusão de *waypoints*. Posteriormente, os *waypoints* podem ser deslocados ao longo da rota, de forma a permitir a alteração desta sem necessitar de criar uma nova rota.
- Durante a navegação (com *GPS* activo), a aplicação possui um mecanismo de orientação automática que conduz o utilizador ao longo da rota através de pictogramas indicadores da direcção a tomar para chegar ao próximo *waypoint*. Quando o utilizador chega a um *waypoint*, pode ser avisado através de imagens ou sons pré-configurados.
- Os *waypoints* da rota podem ser definidos de modo a activar uma *zona de aproximação*. Assim, quando o utilizador está localizado numa zona próxima do *waypoint*, a aplicação pode ser configurada para mostrar instruções de navegação, informação sobre a área a percorrer, avisos, etc.
- Durante a navegação são visualizadas três linhas de navegação. A primeira, indica ao utilizador a rota ideal entre o *waypoint* anterior e o próximo. A segunda, indica ao utilizador a rota a seguir desde o ponto onde se encontra até ao próximo *waypoint* – esta é particularmente útil, no caso do utilizador se encontrar "fora" da rota predefinida. A terceira, é uma linha de projecção de navegação que indica ao utilizador o seu destino segundo a direcção actual.

- As rotas podem ser registadas num ficheiro, podendo ser posteriormente recarregadas, lidas e partilhadas com outros utilizadores.

***Tracks*:**

- Possibilidade de registo de *tracks* através de definição de intervalos temporais ou espaciais.
- Após o registo de um *track*, a aplicação permite a adição de *waypoints* e a divisão/separação do *track* original em vários *tracks* individuais.
- A aplicação também permite o *replay* de *tracks*. O *replay* é acompanhado da indicação constante da velocidade, altitude, localização do utilizador, distância percorrida e tempo decorrido.
- Os *tracks* podem ser registados num ficheiro, podendo ser posteriormente recarregados, lidos e partilhados com outros utilizadores.

Outras:

- Criação de *waypoints* simples (não associados a *tracks* ou rotas) que permitem, por exemplo, localizar pontos de interesse (bastante utilizados principalmente na organização de encontros recreativos ao ar livre ou na organização de eventos relacionados com desportos de aventura).
- Possibilidade de edição de várias propriedades relativas ao *waypoint*: nome, cor, símbolo e associação a um determinado ficheiro.
- O sistema permite a conversão entre rota e *track* e vice-versa.
- Configuração das várias propriedades relacionadas com a interface *GPS* (porta de comunicação, *baudrate*, etc.).
- Visualização do estado do *GPS*: qualidade do sinal recebido, número de satélites activos e respectiva posição.

Prós:

- Grande diversidade de funcionalidades e opções.
- Extrema facilidade de adaptação da aplicação a diversos domínios de utilização (navegação aérea, marítima e terrestre).
- Instalação fácil e rápida.
- Permite utilizar vários tipos de mapas incluíndo mapas do próprio utilizador.

Contras:

- Interface gráfica poderosa, mas complexa e pouco intuitiva.
- Curva de aprendizagem lenta.
- Os mapas precisam de ser convertidos para um formato proprietário por uma aplicação (incluída com o *Ozi Explorer CE*) antes de serem utilizados no *Ozi Exlorer CE*.

Como se pode constatar através das funcionalidades apresentadas, esta aplicação é constituída por uma diversidade de funcionalidades que não está presente no sistema anterior. Consequentemente, este sistema apresenta um nível de complexidade superior, o que, neste caso, resulta numa interface mais confusa e menos intuitiva. Embora esta característica prejudique a usabilidade do sistema e o torne menos apelativo para o utilizador comum que prefere ter todas as funcionalidades à distância de um clique, permite aumentar a flexibilidade da aplicação possibilitando a sua adaptação a vários contextos de utilização [Pocketpcmag, 2005].

A filosofia subjacente ao desenho desta aplicação assenta na premissa de que o utilizador "assíduo" deste tipo de sistemas prefere um maior número de funcionalidades em detrimento de uma curva de aprendizagem mais rápida. Assim, este produto pretende preencher o nicho de mercado relativo aos utilizadores que já têm algum conhecimento na área dos conceitos de navegação e experiência na utilização de mapas topográficos e que pretendem uma aplicação que preencha todos os seus requisitos em termos de navegação [Pocketgps, 2005].

No entanto, existem outras opções no mercado que permitem satisfazer os utilizadores com menor experiência nesta área, mas que pretendem usufruir deste tipo de funcionalidades. Entre essas, destacam-se o *Maptech Outdoor Navigator*, *Memory-Map Navigator* e o *VITO SmartMap*. Este último merece particular atenção, pois possui duas características a salientar relativamente ao sistema apresentado: permite visualizar mapas em formato *raster* e vectorial e tem uma interface gráfica simples e atraente, sendo por isso indicado para os utilizadores menos experientes. Contudo, fica bastante aquém do *Ozi Explorer CE* em termos de funcionalidades disponibilizadas.

Figura 3.2 – *Screenshot* da aplicação *Ozi Explorer CE*.

Apesar dos vários sistemas semelhantes disponíveis, optou-se pela apresentação do *Ozi Explorer CE*, pois é o sistema que melhor representa o conjunto de sistemas existentes nesta área devido às abrangentes funcionalidades que incorpora. Finalmente, importa ainda salientar que, tal como na maioria das restantes aplicações existentes nesta área, o *Ozi Explorer CE* é utilizado conjuntamente com outra aplicação – *Ozi Explorer* – que corre no *PC* e que permite calibrar e exportar os mapas para utilização posterior no *Ozi Exlorer CE*.

3.1.3 Conclusões

Existem várias soluções no mercado que disponibilizam sistemas de navegação através de *GPS*. No entanto, cada solução tem funcionalidades próprias e destina-se a diferentes tipos de utilizadores. Como se pode ver, o *Tom Tom Navigator* é um sistema de navegação rodoviária que possui uma interface gráfica fácil de usar e, por isso, destinada ao utilizador comum. Por outro lado, a aplicação *OziExplorer CE* é uma aplicação destinada à navegação *off-road*, mais complexa e destinada ao utilizador que possui já alguma experiência com mapas topográficos.

No caso do *MapAdventure* pretende-se implementar um sistema de navegação *off-road* e destinado ao utilizador comum. Através da análise realizada às duas aplicações, pode-se concluir que os dois factores relevantes no desenho desta aplicação englobam uma interface gráfica fácil de usar e um conjunto de ferramentas que colmata as necessidades dos utilizadores nesta área.

3.2 A plataforma de desenvolvimento *.NET Framework*

A plataforma *.NET Framework* pode definir-se como uma plataforma de desenvolvimento de *software* que permite desenvolver aplicações de um modo rápido, eficiente e escalável em várias linguagens de programação (*Visual Basic .NET, C#, ASP .NET, JScript .NET*, etc.). A plataforma oferece os seguintes serviços: ferramentas de apoio ao desenvolvimento de aplicações, ambiente que permite a execução de aplicações e *software* que permite aos programadores trabalharem com maior eficiência [Thai et al., 2001].

A arquitectura da *.NET Framework* (Figura 3.1) tem dois componentes principais: o ***Common Language Runtime*** (*CLR*) e a ***.NET Framework class library***. Todas as aplicações escritas em *.NET* partilham o mesmo ambiente, sendo este ambiente constituído pela máquina virtual *CLR* e pela biblioteca de suporte *.NET Framework class library*. A máquina virtual funciona como um "agente" que gere o código em tempo de execução, providenciando serviços essenciais como a gestão de memória, gestão de *threads* e *remoting* e, simultaneamente, assegura a segurança e robustez do código em execução. O código a executar no âmbito do *CLR* é denominado por *managed code*, em contraste com o *unmanaged code* que se refere ao código que corre "fora" do *CLR*.

Um dos principais objectivos da plataforma *.NET* consiste na integração de várias linguagens de programação de tal modo que os programas possam ser escritos em linguagens diferentes e, no entanto, possam interagir entre si. Contudo, cada linguagem de programação tem as suas funcionalidades próprias. De modo a tornar possível a adaptação de diferentes linguagens a esta plataforma, a *Microsoft* especificou uma série de regras básicas descritas na *Common Language Specification* que determinam os requisitos mínimos a que uma linguagem tem que

obedecer de modo a ser compatível com a *.NET Framework*. Assim sendo, os fabricantes de compiladores podem construir compiladores segundo esta especificação permitindo a adaptação das várias linguagens de programação existentes à plataforma *.NET*.

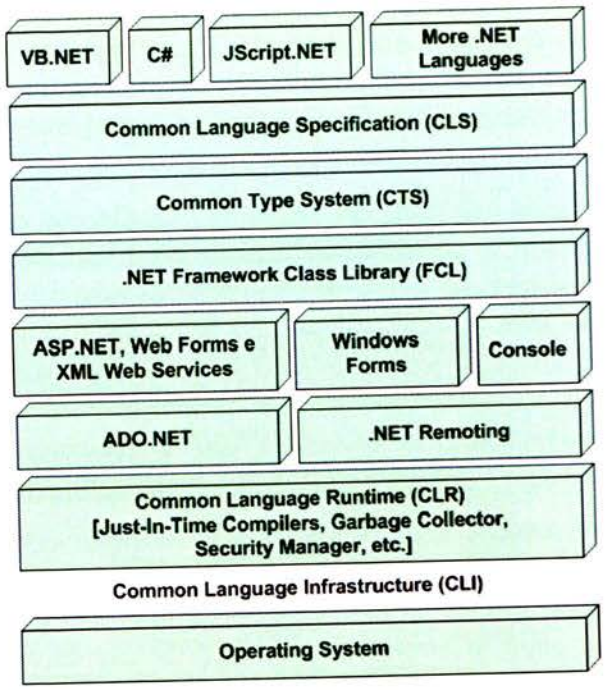

Figura 3.3 – Arquitectura da plataforma *.NET Framework*

A biblioteca de suporte (*.NET Framework class library*) consiste numa colecção de classes reutilizáveis que se baseiam no paradigma de orientação aos objectos. Estas classes podem ser usadas para desenvolver vários tipos de aplicações, desde as aplicações baseadas em interface gráfica (*Windows Forms*), passando pelas aplicações *Web* (*ASP .NET Web Forms*), até aos recentes *Web Services* baseados em *XML*.

3.2.1 *Common Language Runtime*

A principal função do *CLR* [Liberty, 2003] é a tradução da linguagem intermédia designada de *Microsoft Intermediate Language* (*MSIL*) para código nativo através dos compiladores *Just In Time* (*JIT*). No entanto, como se pode ver na figura 3.2, existem outras funcionalidades e mecanismos incluídos no *CLR* que permitem a execução robusta de um programa. Entre estes destacam-se o *class loader*, o *verifier*, e outros mecanismos de suporte à execução de código (*Garbage Collector*, gestão de excepções, suporte de *threads*, etc.).

Quando uma aplicação *.NET* é iniciada num sistema operativo *Windows*, o *loader*[9] do sistema operativo reconhece a aplicação e passa imediatamente o controlo para o *CLR*. Este encontra o ponto de entrada da aplicação, que normalmente está associada à função *Main()*, e passa o controlo de execução para o ***class loader*** que carrega as classes *.NET* para memória e as prepara para execução. No entanto, antes de proceder a esta operação, o *class loader* necessita de encontrar a classe-alvo (*target class*). Para tal, procura informação em diferentes

[9] Sistema que copia o programa para memória principal.

localizações, incluíndo no ficheiro de configuração da aplicação (*.config*), no *GAC*[10], e na meta-informação que constitui o *assembly* (*.EXE* ou *.DLL*[11]). A informação recolhida é crucial, pois permite determinar a localização da classe correcta. Após ter encontrado e carregado a classe-alvo, guarda informação sobre esta, de modo a não ter que a carregar novamente durante o processo de execução. Posteriormente, se a classe-alvo referenciar outras classes estas serão carregadas através do mesmo processo. No entanto, se estas classes já tiverem sido carregadas, o *class loader* não efectua nenhuma operação. Finalmente, o *class loader* usa os metadados apropriados para inicializar os variáveis estáticas e instanciar os objectos das classes em memória.

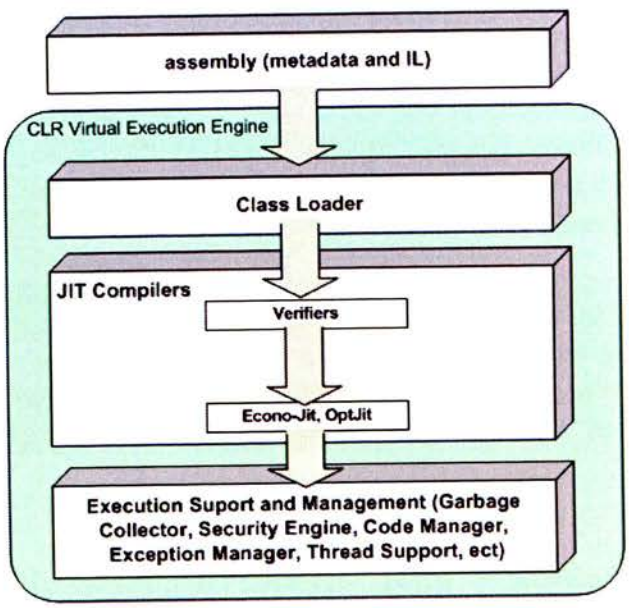

Figura 3.4 – Diagrama de blocos do *CLR*

Depois da classe ter sido carregada e antes da respectiva linguagem intermédia (*MSIL*) ser compilada, o ***verifier*** verifica, em tempo de execução, se o código é seguro e se pode ser executado. O *verifier* é responsável pela verificação da estrutura da meta-informação e assegura que a linguagem intermédia a ser executada é segura. Como o *verifier* faz parte do *JIT*, este só é "activado" quando um método é invocado, o que constitui uma diferença fundamental entre o ambiente *.NET* e outros ambientes. Importa ainda salientar que este passo é opcional. Isto é, se o código for fiável não é verificado, mas redireccionado directamente para o **compilador *JIT***.

Na realidade, o código é sujeito a duas compilações. Na primeira, o código-fonte é compilado através de *managed compilers* (*csc*, *vbc* e outros) que compilam *C#*, *Visual Basic .NET* e outras linguagens *.NET* para gerar um *assembly* (*.EXE* ou *DLL*) que contém meta-informação e linguagem intermédia (*MSIL*). A segunda compilação ocorre em tempo de execução e inclui a verificação do *assembly* pelo *verifier* e a compilação da linguagem intermédia contida no *assembly* para código binário nativo através do compilador *JIT*.

---

[10] Directório onde estão localização as *assemblies* registadas do sistema. Esta ferramenta possibilita a partilha de uma assembly por várias aplicações.

[11] *Dynamic Link Library*

No entanto, a conversão de um programa inteiro para código nativo é ineficiente e pode comprometer o desempenho da aplicação. Assim, por razões de optimização, a compilação de determinado método só ocorre na primeira vez que o método é invocado. Na primeira compilação de um método, o compilador *JIT* guarda o código nativo do método em memória, bem como um "apontador" para este. Durante as invocações subsequentes do método, não ocorre compilação, mas é lida a posição de memória onde se encontra o código nativo. Assim, os métodos não utilizados durante o tempo de execução nunca são compilados, optimizando todo a processo de execução.

Outra das vantagens do compilador *JIT* reside no facto de poder compilar código nativo dinamicamente para uma determinada máquina. Assim, o mesmo *assembly* pode ser executado em máquinas de um processador ou de duplo processador, pois o compilador *JIT* compila e optimiza o código nativo tendo em conta a máquina-alvo. Outra vantagem óbvia é a possibilidade de executar código noutras plataformas (por exemplo, em Unix) desde que estas incluam um *CLR*. Para tal, a *Microsoft* desenvolveu uma especificação denominada por *Commom Language Infrastructure* que possibilita o desenvolvimento de ambientes de execução *.NET* para outras plataformas[12] (*Mac OS X, Linux* e outras).

Além das funcionalidades referidas anteriorente, o *CLR* suporta ainda outros serviços de suporte que serão brevemente descritos.

- **Gestão automática de memória:** Ao contrário da linguagem *C++* onde todos os objectos têm de ser eliminados depois de usados, o *CLR* suporta a gestão automática do tempo de vida dos objectos. O *garbage collector* detecta quando os objectos já não são referenciados e elimina-os, evitando a existência de apontadores "perdidos" ou de referências circulares.

- **Tratamento de excepções:** Antes da *.NET Framework* não existiam mecanismos consistentes de gestão de excepções que permitissem apoiar o tratamento de excepções e/ou erros. Na *.NET Framework*, o *CLR* suporta um mecanismo de gestão de excepções que abrange todas as linguagens.

- **Segurança:** O *CLR* permite assegurar que o código em execução é válido através de vários mecanismos de segurança que actuam em tempo de execução. Estes mecanismos podem ser baseados na origem do código, na identidade do seu utilizador ou em permissões.

- **Depuração:** O *CLR* possui diversas ferramentas de depuração e de análise dinâmica de código (*profiling*). Adicionalmente, a *framework* fornece duas *API's*[13] para o desenvolvimento de ferramentas de depuração e de análise dinâmica de código.

- **Interoperação:** O *CLR* suporta a interoperação entre *managed code* (compilado pelo *CLR*) e *unmanaged code*. A plataforma *.NET* fornece uma funcionalidade denominada *COM Interop* que funciona como "ponte" entre o mundo *COM* e *CLR*, permitindo a um objecto *COM* aceder a um objecto *.NET*. Existe ainda outro mecanismo denominado de *Platform Invoke* (*P/Invoke*) que possibilita realizar chamadas a *unmanaged code* em linguagens *managed* (*C#, VB.NET* e outras).

---

[12] Actualmente, já existem implementações em multi-plataformas (como o *MONO* [MONO, 2005] e o *dotGNU* [DotGNU, 2005]) da infraestrutura *.NET*.

[13] *Application Program Interface*

3.2.2 A Biblioteca de Suporte

Na figura 3.1, a camada denominada *.NET Framework Class Library* refere-se ao conjunto das classes básicas da *framework*. Estas classes permitem o acesso a funções nativas do sistema operativo e disponibilizam funcionalidades "rudimentares" como o acesso a ficheiros e recursos de rede, acesso a dispositivos periféricos, gestão de segurança, manipulação de *strings*, etc. Sobre esta camada existe outro conjunto de classes, denominadas de *ADO.NET*, que extendem a funcionalidade das classes básicas de modo a suportarem gestão adicional de dados. Estas classes incluem suporte para *SQL* e permitem o acesso a bases de dados, mapeamento em memória de modelos relacionais. As classe denominadas *.NET Remoting* permitem a comunicação entre aplicações. Deste modo, os objectos *.NET* são expostos a processos remotos, permitindo a comunicação entre processos. As restantes classes permitem efectuar o desenvolvimento *Web* (*ASP.NET* e *Web Forms*), criar aplicações com interface gráfica no ambiente *Windows* (*Windows Forms*) e desenvolver aplicações distribuídas (*Web Services*).

3.3 A plataforma de desenvolvimento *.NET Compact Framework*

A *.NET Compact Framework* (*.NETCF*) [Wigley et al., 2003] foi desenvolvida com o objectivo de possibilitar o desenvolvimento rápido e eficaz de aplicações para diversas plataformas móveis (*Pocket PC, Smartphone*, entre outros). Esta plataforma, ao contrário da descrita anteriormente, só possui suporte para duas linguagens de programação (*VB.NET e C#*). No entanto, uma das suas grandes vantagens reside na sua integração com o *Microsoft Visual Studio .NET* (*VS .NET*). Deste modo, o programador pode usufruir de funcionalidades como depuração, emulação de dispositivos móveis e *templates* que tornam o trabalho do programador menos moroso e mais eficiente.

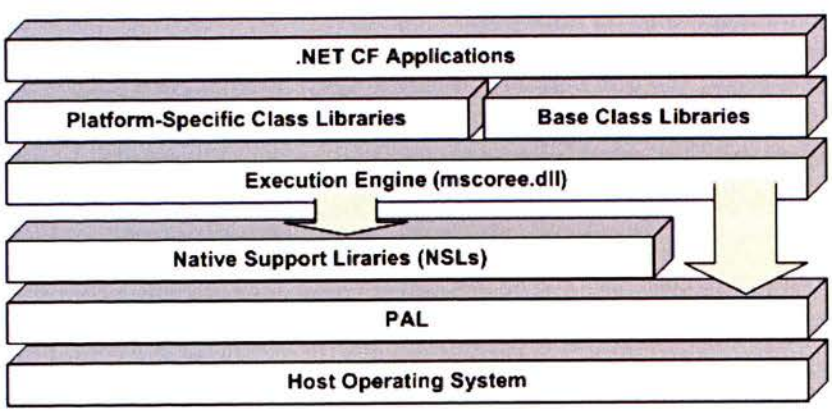

Figura 3.5 – Diagrama de blocos da .NET Compact Framework

Embora esta *framework* possua uma arquitectura idêntica à *.NET Framework*, foi totalmente redesenhada tendo em conta os dispositivos móveis. No entanto, devido às similaridades com a *.NET Framework*, esta secção pretende apresentar apenas as diferenças de maior relevo entre as duas *frameworks*. Na figura 3.3, apresenta-se a arquitectura da plataforma *.NETCF*. A diferenças mais significativas relativamente à *.NET Framework* residem nas camadas

Platform Adaptation Layer (PAL) e *Native Support Libraries (NSL)*, dado que o *Execution Engine (EE)* facilita as mesmas funções que o *CLR* na *.NET Framework.*

A camada de abstracção ***PAL*** é o componente primário que torna a portabilidade da plataforma possível, pois permite a execução de *managed code* em várias sistemas operativos e em diferentes arquitecturas de *CPU* (*ARM, MIPS, SH3, SH4,* etc). Esta camada contém, essencialmente, uma variedade de subsistemas que expõe, através de um conjunto consistente de *API's*, as funcionalidades do sistema operativo e do *hardware* do dispositivo às camadas superiores (*NSL* e *EE*). Esta camada inclui interfaces para a memória do sistema, interrupções, *timers*, portas entrada/saída, entre outros. A camada ***NSL*** consiste num conjunto de bibliotecas que utilizam o *PAL* de modo a fornecer diversas funcionalidades necessárias ao funcionamento da *.NETCF*, incluíndo operações de acesso a ficheiros, globalização, criptografia e operações de manipulação da interface gráfica.

As principais diferenças entre as duas plataformas revelam-se ao nível das bibliotecas de suporte. A *.NETCF* contém apenas um "subconjunto" (cerca de 25%) das classes existentes na biblioteca da *.NET Framework* devido ao menor espaço e capacidade de processamento existente no dispositivo móvel. Assim, e como já se previa, esta plataforma não possui a funcionalidade e a flexibilidade oferecida pela versão *desktop* [Fox et al, 2003].

De seguida, destacam-se as principais omissões existentes na biblioteca de suporte *.NETCF* relativamente à sua correspondente na *.NET Framework.*

- ***ASP.NET***: A *.NETCF* não inclui nenhum suporte para o desenvolvimento *Web*, pois o principal objectivo desta plataforma é o desenvolvimento de aplicações que correm no dispositivo.
- **Acesso *OleDb*:** Esta plataforma omite o espaço de nomes *System.Data.OleDb* e por isso não permite acessos directos à base de dados através do *OleDb .NET Data Provider.*
- **Serialização:** A *.NETCF* apenas permite a serialização de objectos para *XML* de forma a poderem ser usados em *Web Services.*
- ***.NET Remoting*:** Na versão *desktop*, é possível criar aplicações que comunicam entre si através usando objectos "serializados" para binário ou *SOAP*, através dos protocolos *TCP* ou *HTTP*. Esta funcionalidade não está disponível na *.NETCF*, devido ao facto desta *framework* só suportar a serialização de objectos para *XML.*
- **Funcionalidade *XML*:** A *.NETCF* apenas permite ao programador ler e escrever ficheiros *XML*; não disponibiliza funcionalidades como a execução de *queries XPath* ou transformações *XSLT* .
- **Gráficos:** Nesta área foram limitadas algumas das funcionalidades existentes na *.NET Framework.* Assim, a *.NETCF* apenas retém as funcionalidades básicas que permitem a gestão de objectos vectoriais (linhas e polígonos), imagens *raster* e texto, não incluído as funcionalidades gráficas existentes na biblioteca *GDI+*[14] versão *desktop.*

[14] Biblioteca gráfica que contém um conjunto de *API*'s que permitem efectuar a renderização de gráficos 2D, imagens, texto. Adiciona novas funcionalidades e um novo modelo de programação relativamente ao predecessor *GDI.*

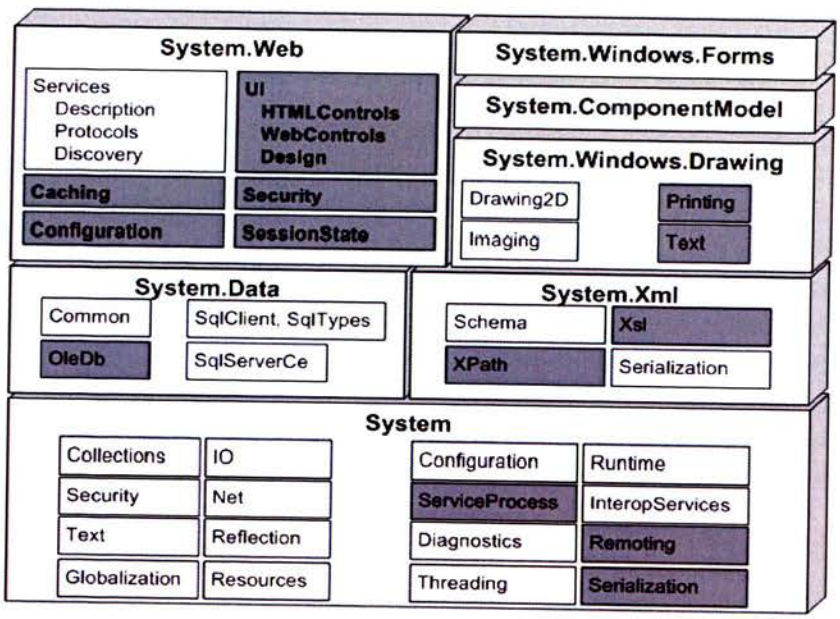

Figura 3.6 – *Namespaces* da *.NET Compact Framework*

Na figura 3.4 apresenta-se uma correspondência entre os *namespaces* existentes na *.NETCF* e na *.NET Framework*. Os *namespaces* salientados existem apenas na versão *desktop*. Deste modo, pretende-se dar uma ideia geral das limitações existentes na *.NETCF*.

3.4 A plataforma de desenvolvimento *eMbedded Visual C++*

Na área das pataforma de desenvolvimento de aplicações para dispositivos móveis importa também salientar a importância do ambiente integrado de desenvolvimento[15] denominado *eMbedded Visual C++*. Embora este *IDE* não tenha sido utilizado no desenvolvimento deste projecto, a sua referência nesta secção justifica-se pelo facto de ser largamente utilizado para o desenvolvimento de aplicações para dispositivos móveis.

Ao contrário do *VS .NET*, este *IDE* é destinado exclusivamente ao desenvolvimento, em código nativo (permite usar as linguagens de programação *C* e *C++*), de aplicações para dispositivos móveis. Tal como o *VS .NET*, disponibiliza ferramentas eficazes de depuração e de apoio ao desenvolvimento. Entre estas, destaca-se o emulador – que permite efectuar, de um modo rápido, testes à aplicação desenvolvida; e a existência de um mecanismo de *just-in-time debugging*[16]. Além destas, contém ainda uma série de ferramentas adicionais bastante úteis (que são uma importante mais-valia face às ferramentas disponibilizadas pelo *VS .NET*):

- *Remote Call Profiler* – Esta ferramenta permite analisar, em tempo de execução, as rotinas chamadas pela aplicação. Assim, à medida que as rotinas são chamadas, o programador pode visualizar o tempo de execução associado a cada rotina, a *thread* em

[15] Do inglês *Integrated Development Environment*.

[16] Esta funcionalidade permite aos programadores gerir excepções não esperadas. Sem este mecanismo a aplicação simplesmente terminaria e o programador teria grande dificuldade em encontrar a origem da excepção.

que executa, os módulos carregados pela aplicação, e outras propriedades configuráveis pelo programador.

- *Remote Kernel Tracker* – Esta ferramenta permite monitorizar remotamente diversas "variáveis" relacionadas com a execução de aplicações no dispositivo móvel: tempo gasto pela aplicação a executar código do *kernel* e do utilizador, operações efectuadas sobre a memória, etc.

- *Remote Performance Monitor* – permite monitorizar remotamente a utilização do *CPU*, a utilização de *threads*, processos, tráfego de rede, bateria, etc.

- Outras ferramentas incluem o *Remote Zoom* que permite capturar o ecrã do dispositivo móvel; o *Remote Heap Walker* que permite visualizar a memória *heap* associada a cada processo; o *Remote Process Viewer* que permite visualizar os processos activos no dispositivo; o *Remote File Viewer* que permite a transferência de ficheiros para o dispositivo; e, finalmente, o *Remote System Information* que permite visualizar informação relativa à configuração do dispositivo móvel (versão do sistema operativo, informação sobre o *CPU*, memória, entre outros).

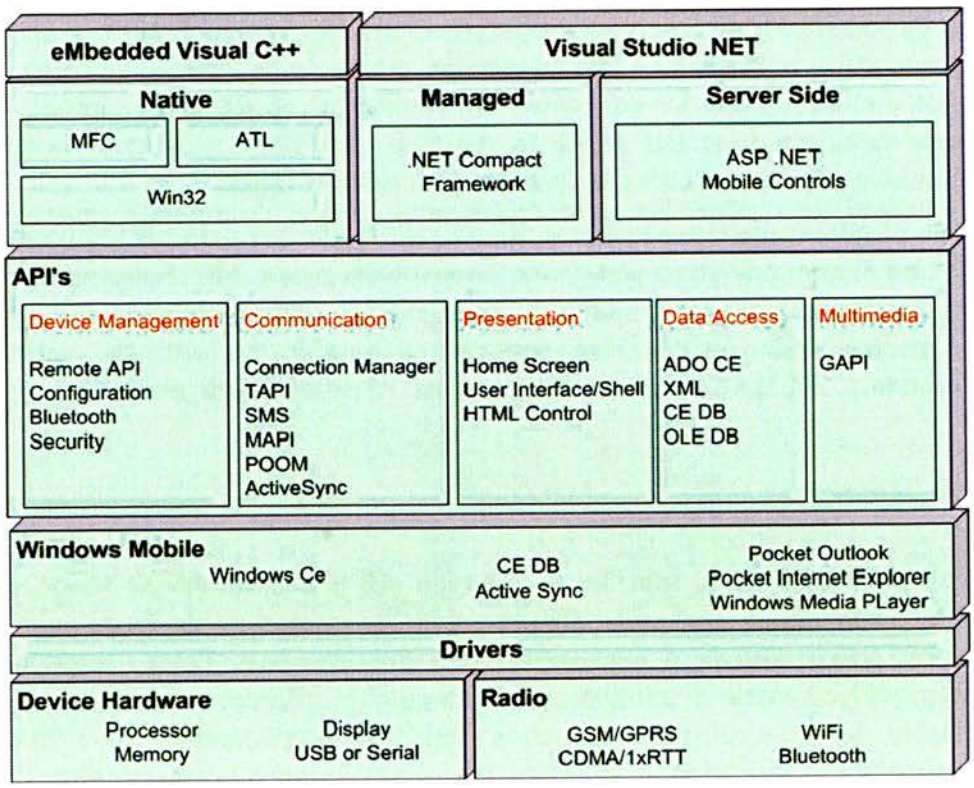

Figura 3.7 – Plataformas de desenvolvimento para dispositivos móveis e *API*'s existentes para os dispositivos *Windows Mobile*[17].

[17] *Windows Mobile* é uma marca criada pela *Microsoft* que identifica simultaneamente *software* incorporado nos dispositivos *Pocket PC* e *SmartPhone*.

Embora o *VS .NET* tenha um motor de depuração bastante eficaz, a inclusão destas ferramentas seria bem-vinda, pois ajudam bastante o programador na árdua tarefa de depuração.

Além destas ferramentas de apoio, existem ainda bibliotecas (*MFC*, *WTL* e *ATL*) que permitem ao programador desenvolver aplicações através de *templates*, tornando o ciclo de desenvolvimento mais rápido. Contudo, é necessário ter alguma experiência e conhecimento teórico destas bibliotecas para o programador poder usufruir plenamente das suas capacidades.

3.4.1 Ferramentas de desenvolvimento

Na realização deste projecto foram utilizadas diversas ferramentas de desenvolvimento e de apoio. Como ambiente integrado de desenvolvimento foi utilizado o *Microsoft Visual Studio .NET*. Esta ferramenta é bastante eficaz, pois permite o rápido desenvolvimento de código, principalmente no que diz respeito ao desenvolvimento da interface gráfica. Esta eficiência deve-se à existência de uma ferramenta denominada *Designer* que permite a rápida edição gráfica da interface das aplicações e que disponibiliza ainda outras funcionalidades importantes como ferramentas eficazes de depuração, *code-completion* e documentação *in-line*. No campo dos dispositivos móveis, possui uma ferramenta de apoio ao desenvolvimento bastante útil – o emulador – que permite ao programador testar a aplicação, de um modo rápido e eficiente, sem ter de recorrer constantemente ao dispositivo móvel. No entanto, a grande vantagem deste *IDE* reside na sua fácil usabilidade, pois todas as ferramentas são intuitivas e exigem uma curva de aprendizagem bastante rápida.

Como plataforma de testes unitários foi utilizado o *Nunit Framework* [Newkirk et al., 2005]. O *NUnit Framework* é uma plataforma desenvolvida para a *.NET Framework*, que permite ao programador elaborar um conjunto de procedimentos unitários que visam testar a integridade do produto desenvolvido. Para controlo de versões foi utilizada a tecnologia *CVS* [Vesperman, 2003] e o *Tortoise CVS* [TortoiseCVS, 2005] como cliente.

3.5 Resumo

Existem várias tipos de soluções no mercado que pretendem atingir vários mercados-alvo. Deste modo, foram analisadas algumas soluções existentes, de forma a tirar conclusões práticas para o sistema a implementar pela *ParadigmaXis*. Foi analisado um sistema de navegação rodoviário e um sistema de navegação *off-road*. Através da análise das duas soluções foi possível concluir que a aplicação *MapAdventure* deve disponibilizar as funcionalidades essenciais ao utilizador, tendo como prioridade uma interface simples de usar.

A *.NET Framework* é uma nova plataforma que permite o desenvolvimento integrado de aplicações para prestação de serviços, que reúnem informação de, e interagem com, uma variedade diversificada de fontes, independentemente das plataformas ou linguagens usadas. As linguagens de programação *.NET* (C#, *VB .NET, Jscript .NET,* etc) não fazem sentido sem a utilização da biblioteca de suporte que é incluída na *.NET Framework*. Esta permite o acesso rápido e fácil a várias tecnologias: *Web Services*, acesso a base de dados (*ADO .NET*) e desenvolvimento *Web* (*ASP .NET*). Todas as aplicações escritas para a *.NET Framework* correm dentro de uma máquina virtual chamada *Common Language Runtime* e são descritas

numa linguagem intermédia (*IL*). Isto permite que a *.NET Framework* seja independente da linguagem de programação, desde que esta esteja de acordo com a norma *Common Language Specification.*

A plataforma *.NET Compact Framework* apenas disponibiliza um pequeno conjunto das funcionalidades oferecidas pela *.NET Framework* devido ao pouco espaço disponível e à inferior capacidade de processamento do dispositivo móvel. No entanto, possui grandes potencialidades e permite o rápido desenvolvimento de aplicações para dispositivos móveis. Tal como na plataforma *.NET Framework* também aqui todas as aplicações correm numa máquina virtual semelhante ao *CLR.*

O ambiente integrado de desenvolvimento *Microsoft Visual Studio .NET* disponibiliza ao programador um conjunto de ferramentas que permitem o rápido desenvolvimento de aplicações. Entre estas, destacam-se as ferramentas de depuração e o emulador. Por seu lado, o ambiente integrado de desenvolvimento *eMbedded Visual C++* permite ao programador desenvolver aplicações, em código nativo, para dispositivos móveis. Este *IDE* possui também um emulador e outras ferramentas úteis que melhoram o processo de depuração e que facilitam o acesso ao dispositivo móvel.

Após a análise destes dois *IDE*'s, a seguinte questão torna-se pertinente: quando usar código nativo e quando usar *managed code*?

De um modo geral, pode afirmar-se que o código nativo é aconselhado para: o desenvolvimento de *drivers*[18] (para dispositivos móveis), objectos *COM* e *ActiveX*, comunicação através de portas série, alterações de elementos do sistema (*Today Screen*, *Input Panel*, etc).

Por outro lado, o *managed code* é aconselhado para: aplicações com bastante interface gráfica, clientes de *Web Services*, clientes de base de dados (*ADO.NET*).

[18] *Software* que permite controlar dispositivos de *hardware* específicos.

Capítulo 4

Especificação e Arquitectura Geral

Este capítulo pretende apresentar a especificação inicial para o sistema a implementar e a descrição da respectiva arquitectura geral. O capítulo inicia-se com uma descrição dos requisitos definidos na fase inicial do estágio. De seguida, é apresentada uma descrição da arquitectura genérica do sistema. A descrição inicia-se com uma apresentação da arquitectura comum às duas aplicações; posteriormente, é feita uma descrição pormenorizada das particularidades das duas aplicações.

4.1 Requisitos

Com base no conhecimento e experiência existente na *ParadigmaXis* e tendo em conta os factores condicionantes do projecto referidos anteriormente, foi então idealizada uma especificação inicial do sistema. De seguida, são descritos os requisitos das duas aplicações que o constituem. Em alguns casos são ainda indicadas as soluções propostas para algumas questões relacionadas com a implementação e arquitectura geral.

4.1.1 *MapAdventure Studio*

Os requisitos funcionais determinados para a aplicação *MapAdventure Studio* são:

Cartografia vectorial:

- **Captura da informação:**
  - Eixos de via: geometria, sentido de trânsito, conectividade, restrições e observações;
  - Localidades: pontos indicativos das localidades, organizados em hierarquia;
  - Estilos: definição dos estilos de visualização dos três tipos de informação;
- **Visualização:**

- Facilidades comuns de navegação: *zoom in/out/all, zoom to selection/fit* e *pan up/down/left/right*;
- Visualização da informação usando os estilos associados;
- Pesquisa alfanumérica nas vias e localidades;
- Interacção entre os resultados da pesquisa e o mapa;

Cartografia militar:

A cartografia *raster* a visualizar é o conjunto de cartas militares do Instituto Geográfico do Exército (IGEOE), correspondentes à Série M888 (escala 1:25.000).

- **O formato da informação:**
  - São 626 cartas, ocupando cada uma delas cerca de 25 *MB*, em formato *geoTiff*[19].
  - As imagens a visualizar estão num formato proprietário da *ParadigmaXis*, obtidas a partir das cartas do IGEOE.
  - Relativos a cada carta, existem vários conjuntos de imagens, para permitir melhor qualidade e maior eficiência independentemente do nível de *zoom* em que vão ser visualizadas.
  - A informação é acedida através de uma tabela de indexação.
  - As imagens são lidas e libertadas usando a tabela de indexação segundo as necessidades.
- **Visualização**
  - Usando a tabela de indexação, são carregadas as imagens necessárias e suficientes para a área e nível de zoom considerados.
  - As imagens que deixam de ser necessárias são libertadas.

Interface *MapAdventure*

Este módulo permite exportar a informação seleccionada pelo utilizador para o dispositivo móvel de modo a poder ser utilizada pela aplicação *MapAdventure*. De um modo mais concreto, este conjunto de funcionalidades visa seleccionar e preparar um conjunto de mapas militares para usar no dispositivo móvel:

- **Quadriculado representativo das cartas militares:**
  - Mostrar a grelha total da Série M888, com identificação de cada uma das cartas.
  - Realçar o conjunto de cartas existentes localmente (o utilizador pode ter adquirido apenas um conjunto; por exemplo, zona norte).
  - Pesquisa da carta militar pela sua designação.
  - Interacção entre o mapa e os resultados da pesquisa.

[19] Tipo de ficheiro que possibilita a inclusão de informação geográfica juntamente com a imagem.

- **Selecção das imagens a usar no dispositivo móvel:**
  - Permitir a escolha por definição de múltiplas áreas, não necessariamente contíguas.
  - Permitir a selecção por carta militar ou por região definida pelo utilizador.
  - Indicação permanente do espaço necessário para cada região seleccionada e para o total das cartas seleccionadas.
  - As imagens a exportar devem ser encriptadas de modo a não permitir o acesso ilegal a estas. Deste modo, as imagens apenas podem ser acedidas usando a aplicação *Map Adventure*.

4.1.2 *MapAdventure*

Os requisitos funcionais determinados para a aplicação *MapAdventure* são:

Cartografia vectorial

- **Captura da informação:**
  - Polígonos: informação diversa relativa a hidrografia, altimetria, limites administrativos, etc;

A restante cartografia vectorial é igual à existente no *MapAdventure Studio*.

Cartografia militar:

Esta secção também herda os requisitos da aplicação anterior.

Global Positioning System:

- Esta funcionalidade permite ao utilizador activar o *GPS* a partir da aplicação;
- Configuração da ligação com o *GPS* (porta de comunicação, *baudrate*, *stop bits*, *paridade* e *data bits*).
- Informação do estado do *GPS*: número de satélites activos, potência do sinal recebido, bússola digital, coordenadas da posição do utilizador no formato *WGS 84*[20], altitude e velocidade.
- Opção de sincronização do mapa com a posição do utilizador (*Follow Me*).
- Definição da posição do utilizador no mapa através de coordenadas *WGS 84*.

[20] Sistema de coordenadas que permite definir as coordenadas na superfície terrestre através de longitude e latitude.

Tracking

- Esta funcionalidade permite ao utilizador construir um registo geográfico do percurso percorrido durante um determinado intervalo de tempo e guardar os dados num ficheiro. Posteriormente, o utilizador pode aceder ao ficheiro noutro dispositivo e aceder ao percurso.
- Possibilidade de abrir e visualizar várias *tracks*.
- Os *waypoints* podem ser definidos através de um **intervalo de tempo** definido pelo utilizador. Deste modo, o utilizador pode definir o nível de pormenor no registo do percurso com base no tempo decorrido entre cada *waypoint*.
- Os *waypoints* podem também ser definidos através de **intervalos de distâncias percorridas** de forma a permitir ao utilizador a definição do nível de pormenor do registo das coordenadas com base na distância entre cada *waypoint*.

Contadores

- Visualização de vários contadores: conta-quilómetros total e parcial; cronómetro total, parcial e parado.
- Opção de *reset* indívidual aos contadores.
- Opção de ajuste manual dos conta-quilómetros.

Nota: As funcionalidades como o conta-quilómetros, *tracking* e a visualização do estado do *GPS* só estão disponíveis no caso do utilizador ter o *GPS* activo.

Foram ainda traçados os seguintes requisitos não funcionais:

- Na aplicação *MapAdventure Studio* as operações de selecção e exportação devem ser de fácil identificação, realizadas através de um número mínimo de passos e sem ambiguidades.
- Na aplicação *MapAdventure*, a interface gráfica deve ser projectada tendo em conta o dispositivo móvel. A interface deve incluir menus simples e fáceis de navegar que permitam aceder de um modo rápido a todas as funcionalidades. As funcionalidades devem ser acedidas através do menor número de passos, de modo a possibilitar o uso da aplicação em ambientes adversos que limitem a interactividade do utilizador com o dispositivo. No mesmo sentido, os ícones devem ser explícitos e de dimensões consideráveis de modo a possibilitar o acesso às várias funcionalidades através do contacto táctil. Assim sendo, toda a interface deve ser projectada segundo a filosofia do "dedo grosso"[21].

[21] Técnica de desenho da interface gráfica de aplicações de dispositivos móveis que possuem um monitor sensível ao toque que permite o acesso às várias funcionalidades da aplicação através do toque do dedo.

4.2 Arquitectura Lógica

Nesta secção apresenta-se a arquitectura lógica do sistema a implementar, nomeadamente no que respeita à composição comum às duas Nas subsecções seguintes apresentam-se os pormenores relativos à arquitectura lógica de cada aplicação em particular.

A principal dificuldade encontrada na construção desta arquitectura centrou-se na necessidade de satisfazer os dois requisitos iniciais que foram definidos para a sua elaboração:

- a arquitectura deveria ser projectada de modo a permitir a sua utilização pelas duas aplicações;
- A arquitectura deveria ser construída tendo em conta posteriores evoluções.

A solução encontrada baseia-se na modularização, pois permite a fácil adaptação da arquitectura a posteriores evoluções já que a alteração de um módulo não implica a alteração dos restantes. Ou seja, permite isolar as possíveis alterações à arquitectura, tornando-a mais robusta e flexível. Consequentemente, permite também que a adaptação da arquitectura às duas aplicações possa ocorrer apenas nos módulos necessários sem exigir a alteração de toda a arquitectura. Esta aproximação permite assim conjugar de uma forma elegante ambos os requisitos descritos anteriormente.

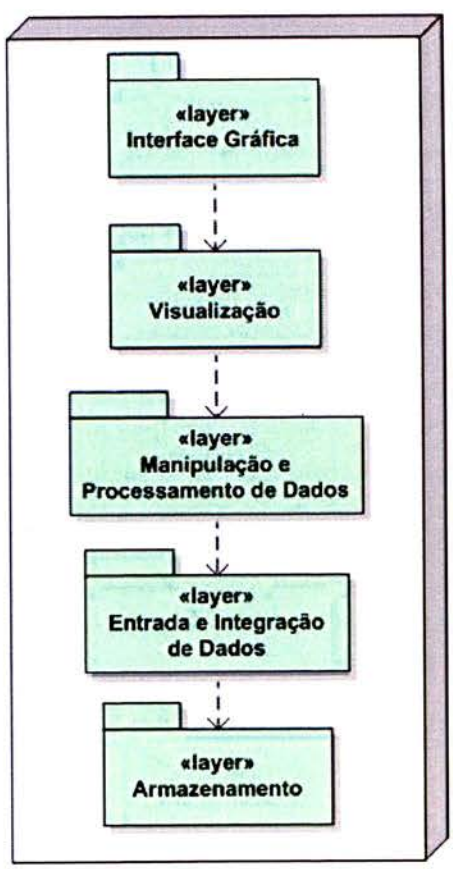

Figura 4.1 – Arquitectura Lógica do Sistema

Assim sendo, foi projectada a arquitectura representada na figura 4.1. Esta estrutura permite que o desenvolvimento das duas aplicações possa ser realizado seguindo o mesmo modelo, mas com diferentes aproximações para cada uma das aplicações. De modo a preservar o conceito da arquitectura, os diversos módulos lógicos devem possuir, genericamente, as mesmas responsabilidades em ambas as aplicações.

De modo a atingir os objectivos pretendidos foi feita uma separação da arquitectura em cinco camadas distintas:

A camada *Armazenamento* inclui apenas os ficheiros onde estão armazenados os dados necessários à construção da cartografia vectorial (eixos de via, localidades, polígonos, estilos associados). Os dados vectoriais e *raster* estão em formatos proprietários da *ParadigmaXis*. Os ficheiros estão embebidos na aplicação de modo a não serem acedidos pelo utilizador.

A camada seguinte – *Entrada e Integração de Dados* – contempla as classes e os métodos que tratam de ler e transferir os dados residentes nos ficheiros para as estruturas de dados apropriadas de modo a serem manipuladas pela camada superior. Assim, quando a aplicação é iniciada são lidos os ficheiros que contêm os dados vectoriais e inseridos nas estruturas de dados apropriados. Posteriormente, a aplicação apenas manipula os dados existentes nas estruturas de dados. As estruturas de dados são constituídas por classes e estruturas que contêm a informação vectorial. Neste contexto importa salientar que ao contrário da cartografia vectorial, a cartografia *raster* não é guardada em estruturas de dados, mas acedida apenas quando necessário, de modo a evitar gastos desnecessários de memória.

A camada denominada por *Manipulação e Processamento de Dados* efectua operações sobre os dados vectoriais e *raster*. Esta é uma das camadas mais importantes, pois é aqui que são implementadas todas funcionalidades que envolvem a gestão da cartografia a ser visualizada no mapa. Esta camada contém informação sobre o estado actual do mapa e efectua a gestão do desenho do mapa. Além desta informação, possui os métodos que são utilizados na manipulação do mapa e que permitem disponibilizar as diversas funcionalidades de navegação do mapa (*zoom in/out, pan down/left/up/right, zoom to selection e zoom to fit*). Esta camada contempla ainda a gestão dos dados *raster* que possibilita a visualização das cartas militares.

A camada *Visualização* refere-se aos pormenores relativos aos métodos gráficos de desenho do mapa. Neste campo, são abrangidas duas técnicas utilizadas para optimizar o desenho do mapa, o *double buffer* e a paralelização do desenho do mapa através da utilização de camadas (não confundir estas camadas com as referidas atrás).

A última camada – *Interface Gráfica* – refere-se aos componentes da interface gráfica. Estes são analisados com maior detalhe em secções seguintes.

4.2.1 Sub-sistemas Funcionais

Na figura 4.2, são apresentados os principais sub-sistemas funcionais comuns a ambas as aplicações do sistema. A figura pretende descrever as interdependências entre os diversos módulos lógicos. Cada módulo corresponde a um agrupamento de classes que têm responsabilidades comuns. Deste modo, é possível efectuar uma descrição das funções que cada sub-sistema assume no contexto da execução.

O sub-sistema *Map Engine* é o elemento central da aplicação pois é este módulo que contém o estado do mapa e que disponibiliza as principais funcionalidades de navegação sobre o mapa. Este módulo recebe pedidos de desenho do *Draw Manager* e processa as operações efectuadas pelo utilizador sobre o mapa enviadas pelo módulo *Map Window*. O pedido de desenho é composto por duas acções sequenciais efectuadas pelos sub-sistemas que manipulam a cartografia vectorial e *raster*:

- Cálculo da cartografia a desenhar;
- e execução dos métodos de desenho relativos a cada tipo de cartografia.

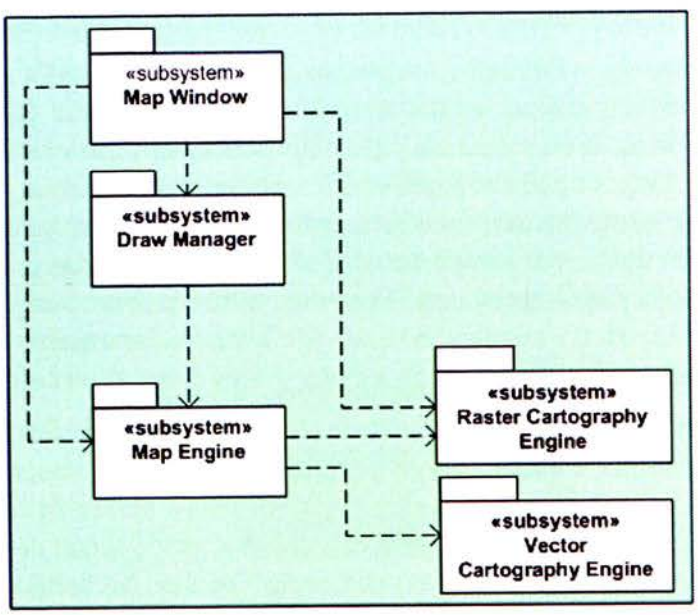

Figura 4.2 – Diagrama de sub-sistemas funcionais comuns às duas aplicações

O *Map Engine* possui toda a informação sobre o mapa e gere todas operações de desenho efectuadas sobre o mapa.

O sub-sistema *Draw Manager* é responsável pelos pedidos de desenho do mapa recebidos do *Map Window* e implementa métodos de optimização de desenho do mapa. Este módulo é importante, pois permite que futuras optimizações aos processos de desenho do mapa possam ser efectuadas sem necessidade de alteração da estrutura do *Map Engine*. Assim, este módulo funciona como um filtro de optimização gráfico entre o *Map Window* e o *Map Engine*.

O sub-sistema *Vector Cartography Engine* processa os pedidos de desenho do *Map Engine* através de métodos de desenho associados a cada tipo de cartografia vectorial (vias e localidades). Este módulo contém ainda várias classes que permitem o desenho da toponímia. Este sub-sistema é bastante flexível, o que permite que cartografia adicional seja adicionada a este módulo sem alteração dos restantes componentes do módulo. Este módulo é também responsável por ler a informação vectorial dos ficheiros para as estruturas de dados apropriadas.

O sub-sistema *Raster Engine* é responsável pela manipulação dos dados *raster*. A gestão dos dados *raster* é conseguida através de um algoritmo multi-resolução que utiliza uma tabela de indexação e informação relativa ao mapa (nível de *zoom* e área a visualizar) para obter as

imagens de melhor qualidade que melhor se adequam à área a visualizar. Assim, as imagens são carregadas quando o mapa é desenhado e são libertadas quando deixam de ser necessárias, permitindo ter em memória apenas as imagens que estão a ser visualizadas. Este módulo é ainda responsável pelos métodos de encriptação da cartografia *raster*.

O sub-sistema *MapWindow* contém os componentes relativas à interface gráfica. Os pormenores relativos a esta área serão descritos com maior pormenor nas secções seguintes, com ênfase nas principais diferenças a nível funcional entre as duas aplicações.

4.2.2 *MapAdventure Studio*

Módulo *Regions*

A principal diferença, a nível funcional, desta aplicação relativamente à estrutura comum apresentada na secção anterior assenta na adição de uma interface de comunicação com a aplicação *MapAdventure*. Esta interface tem duas funções principais: mostrar a grelha das várias cartas militares conjuntamente com o mapa; e disponibilizar as ferramentas e as estruturas necessárias para a selecção e exportação de informação *raster* para o *Pocket PC*. A única consequência, a nível estrutural, foi a adição de um novo módulo – *Regions* – como se pode ver na figura 4.3.

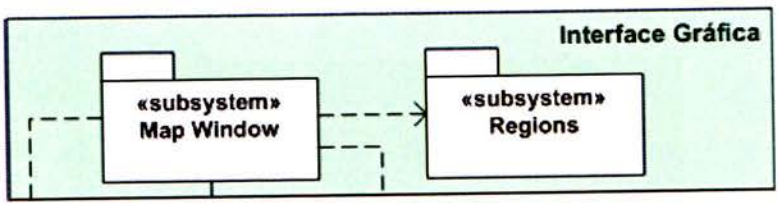

Figura 4.3 – Sub-sistema *Regions*

Este novo módulo tem a responsabilidade de gerir a informação relativa às várias regiões seleccionadas. Cada região corresponde a uma área seleccionada, no mapa, pelo utilizador. A informação relativa às regiões inclui as coordenadas que definem cada região, o espaço utilizado por cada região, o espaço total ocupado por todas as regiões, pormenores da interface gráfica associada a cada região e as cartas militares abrangidas por cada região seleccionada.

A adição da interface de comunicação também teve consequências ao nível de interacção com outros módulos lógicos. Por exemplo, quando o utilizador adiciona ou apaga uma região, o *Map Window* deve ser configurado de forma a enviar a informação respectiva para o módulo *Regions* de modo a permitir a actualização deste. No mesmo sentido, quando o utilizador pretende guardar as regiões seleccionadas, o módulo *Map Window* deve aceder ao módulo *Regions*, obter a informação sobre as regiões guardadas e enviar essa informação ao *Raster Engine* que trata de encriptar e exportar a informação.

A inclusão da grelha de identificação das cartas militares foi facilitada, pois foi perspectivada como se se tratasse de cartografia vectorial. Esta aproximação implicou apenas a adição de novos métodos de acesso a dados e a adição de novas rotinas de desenho no módulo *Vector Engine*, o que é facilitado pela elevada flexibilidade oferecida pela arquitectura definida. Os

módulos *Raster Engine* e *Map Window* foram ainda modificados de modo a possibilitar a identificação das regiões que foram adquiridas pelo utilizador.

Interface Gráfica

Uma grande diferença entre ambas as aplicações é relativa à interface gráfica. Na aplicação *MapAdventure Studio* pretende-se uma interface gráfica minimalista que permita ao utilizador identificar e utilizar facilmente as principais funcionalidades da aplicação – navegação e selecção.

A interface gráfica divide-se em três áreas principais:

- Painel que contém o mapa.
- Barra de ferramentas que contém funcionalidades de navegação e selecção sobre o mapa.
- Painel que disponibiliza informação sobre as regiões seleccionadas e que permite efectuar a pesquisa sobre vias, localidades e cartas militares.

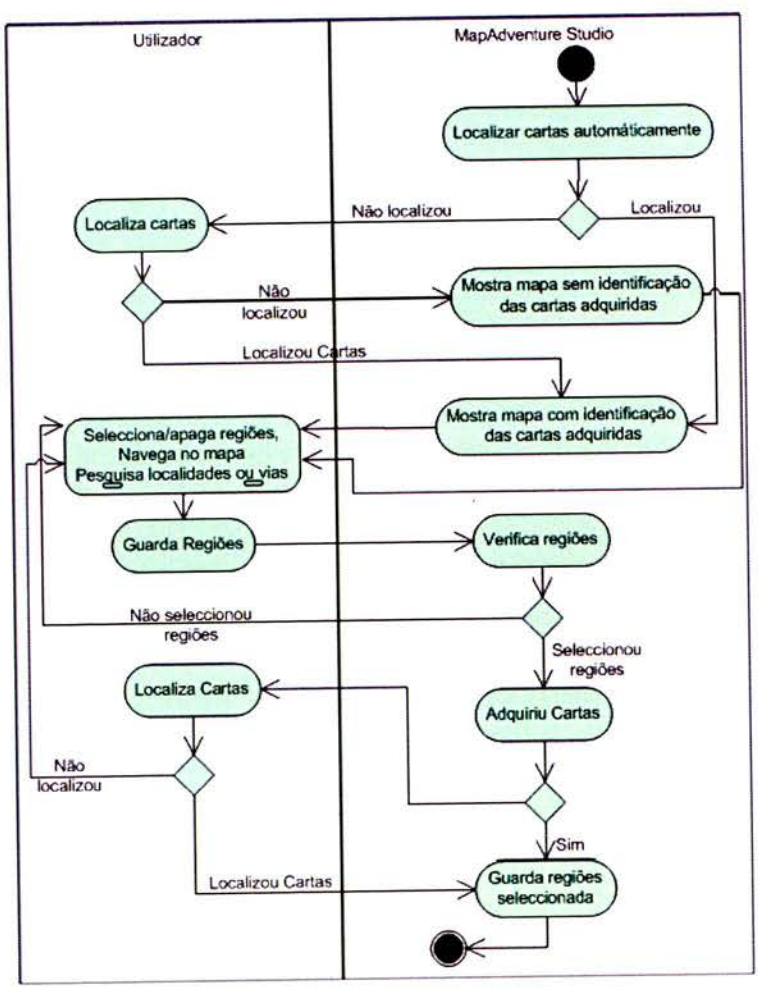

Figura 4.4 – Diagrama de actividades da aplicação *MapAdventure Studio*

Devido à simplicidade exigida à interface gráfica optou-se por apresentar um diagrama de actividades [Rumbaugh et al., 1999] (Figura 4.4) que representa a dinâmica da interacção entre o utilizador e a aplicação. No diagrama de actividades, existe um estado de sub-actividade, o que indica que o estado é composto por várias actividades de navegação e selecção. Este estado corresponde à selecção e pesquisa de regiões. No entanto, não está no âmbito deste diagrama a descrição de tais actividades, dado que apenas pretende descrever a sequência de acções necessárias que permitem ao utilizador exportar as regiões seleccionadas. Analisando a área do diagrama que se refere ao utilizador, é possível verificar que a operação de exportação exige poucos passos intermédios: o utilizador apenas necessita de localizar as cartas que adquiriu, para exportar a informação *raster*. Uma análise mais profunda permite ainda concluir que a aplicação apenas permite que as regiões sejam guardadas se as cartas adquiridas forem localizadas. Em caso contrário, o utilizador apenas pode navegar no mapa e seleccionar regiões.

4.2.3 *MapAdventure*

Na aplicação *MapAdventure* foram integrados dois novos módulos funcionais, o *Track Engine* e o *GPS Engine*, de acordo com a filosofia seguida até ao momento, isto é, permitir que a arquitectura se mantenha flexível e modular de modo a facilitar a sua posterior evolução. Na figura 4.5, apresenta-se um esquema da interdependência entre os módulos funcionais que constituem a aplicação.

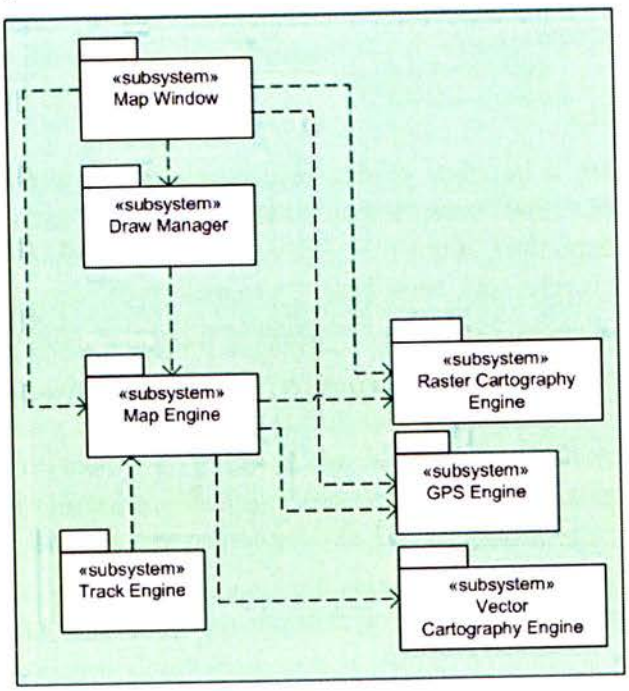

Figura 4.5 – Sub-sistemas funcionais da aplicação *MapAdventure*.

Como é possível verificar, na figura 4.5, o tronco comum da arquitectura não foi alterado com a inclusão dos dois módulos. Estes apenas se interligaram com os restantes. No entanto poderia-se ter adoptado outra solução no caso do *Track Engine*, baseada na sua inclusão no *Vector Cartography Engine*. Esta solução seria viável, pois a estrutura usada para o desenho da cartografia vectorial poderia ser usada para o desenho das *tracks*. No entanto, prevê-se que

em versões futuras sejam adicionadas novas funcionalidades a este módulo, o que justifica a sua individualização.

Como seria de esperar, o módulo *GPS Engine* implementa várias funções importantes, sendo que uma das mais relevantes assenta na leitura de informação que provém do dispositivo *GPS*. Esta operação é composta por dois passos principais:

- Aceder ao dispositivo através de uma porta série virtual emulada pelo sistema operativo.
- Iinterpretar e validar a informação recebida do dispositivo *GPS*.

A leitura da informação é realizada através de um *parser* que descodifica e valida a informação *GPS* (altitude, número de satélites activos no dispositivo, coordenadas, velocidade, nível de sinal recebido). Posteriormente, esta informação é enviada para o módulo *Map Window* e para o *Map Engine*. Deste modo, o *Map Window* pode disponibilizar essa informação ao utilizador.

O módulo *Track Engine* tem como funcionalidades principais a leitura dos ficheiros de *tracks* que estão em formato proprietário da *ParadigmaXis* e o fornecimento da informação relativa aos *tracks* e respectivos *waypoints* ao *Map Engine*. Este, por sua vez, usa esta informação para os incluir no mapa juntamente com a cartografia vectorial e *raster*. Além destas funcionalidades, possibilita o registo de *tracks* (através da distância percorrida ou tempo decorrido) com base em informação recebida do *Map Engine*, nomeadamente as coordenadas do utilizador recebidas do *GPS*.

Interface Gráfica

No *MapAdventure*, a interface gráfica reveste-se de uma importância acrescida devido ao facto de a aplicação correr num dispositivo móvel. Uma das maiores limitações do dispositivo é a área gráfica disponível, já que o ecrã possui dimensões relativamente reduzidas (240x320). A dificuldade mais relevante, neste caso, é a representação da informação sem comprometer a usabilidade da aplicação. Para tal, foi seguida uma filosofia baseada nas seguintes premissas:

- Automatizar os processos – a interface deve permitir o acesso às várias funcionalidades de modo idêntico;
- Mostrar apenas as funcionalidades mais relevantes de modo a evitar o excesso de informação numa janela;
- Agrupar funcionalidades relacionadas de modo a permitir a sua fácil localização;
- Reduzir o número de passos e a quantidade de esforço (por exemplo: introduzir dados) necessários para completar uma tarefa;
- Permitir a fácil identificação das funcionalidades através de ícones elucidativos.

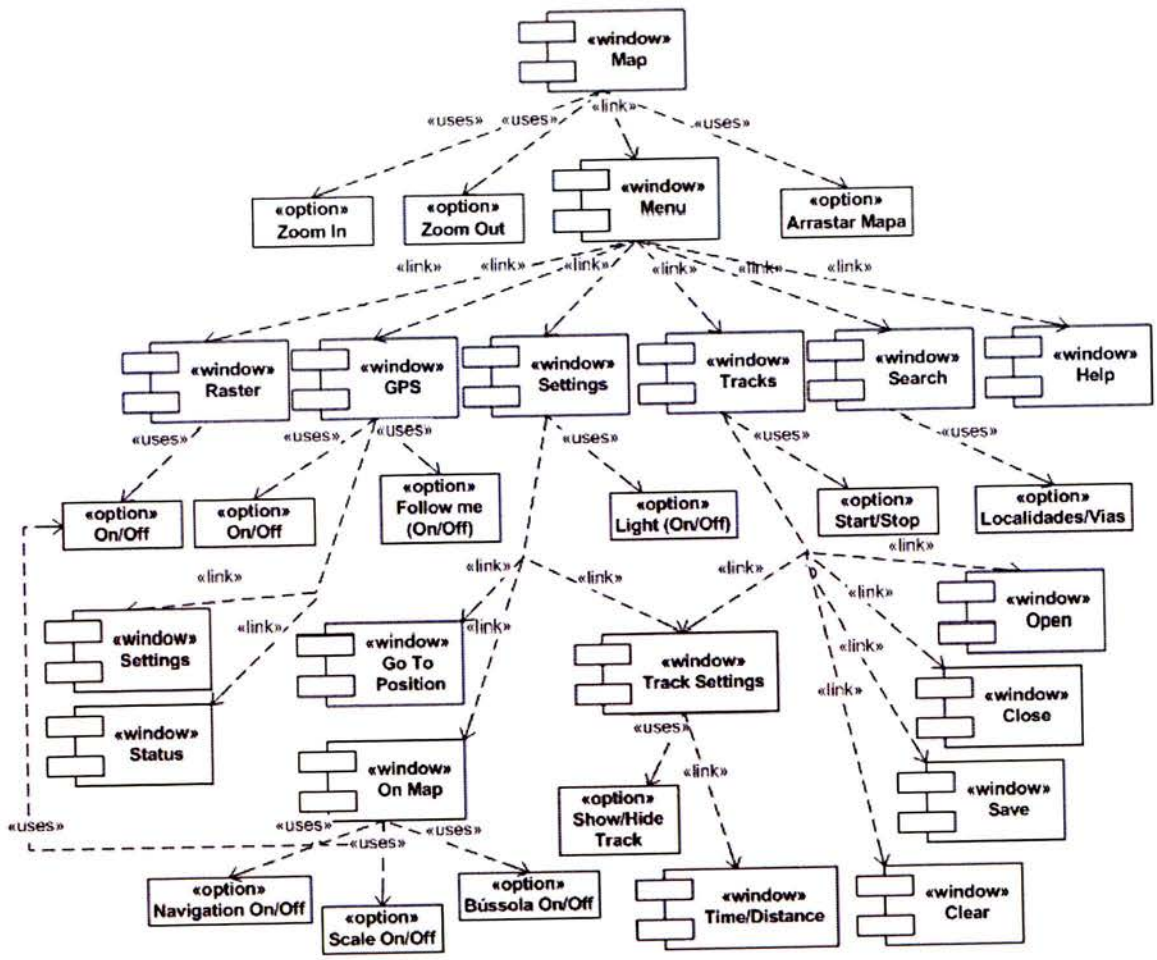

Figura 4.6 – Diagrama de componentes da interface gráfica do *MapAdventure*

Como já foi referido anteriomente, o dispositivo móvel possui uma reduzida capacidade de memória e de processamento. Assim, a interface não deve ser muito "pesada" de modo a evitar longos tempos de inicialização e níveis de desempenho indesejáveis. Neste contexto, o termo pesado indica que a implementação da interface gráfica não deve implicar a criação de demasiados objectos e/ou métodos, devido ao *overhead* associado com a alocação de objectos em memória. Além de que, a criação de demasiados objectos implica também gastos acrescidos de memória que afectam negativamente o desempenho da aplicação.

De modo a atingir os objectivos enunciados anteriormente, foi elaborado o projecto de interface gráfica apresentado na figura 4.6. Foi utilizado um diagrama de componentes de modo a permitir compreender, simultaneamente, a interacção entre os vários componentes e o modo como estes estão organizados sequencialmente. Os componentes designados por *window* representam janelas que contêm várias funcionalidades e/ou ligações para outras janelas (*link*). Os componentes designados por *option* indicam que o utilizador pode efectuar diversas operações, como introduzir e/ou visualizar dados, activar/desactivar funcionalidades (*option*), etc.

Este modelo revela uma grande utilidade, pois actua como uma eficiente ferramenta de análise da usabilidade da interface gráfica. A primeira conclusão que se pode retirar da análise do diagrama é que as funcionalidades são facilmente acessíveis: para aceder a qualquer opção

são necessários, no máximo, três "cliques", o que é bastante eficiente. Observando o diagrama e seguindo as várias sequências possíveis, verifica-se também que o modo de aceder às várias funcionalidades é idêntico, o que facilita a adaptação do utilizador à aplicação e facilita a memorização de todo o processo de interacção com a interface gráfica. A aplicação permite ainda que as funcionalidades mais utilizadas possam ser acedidas mais rapidamente. No caso particular desta aplicação, estas características têm uma importância acrescida, pois a sua utilização destina-se principalmente a ambientes onde o utilizador pode não ter muito tempo para dedicar à sua manipulação.

Nesta figura foram omitidas duas funcionalidades de modo a evitar a complexidade do diagrama:

- Todos os componentes designados por *window* que pertencem ao *menu* contêm um *link* para o componente designado por *Map*;
- Todos os componentes designados por *window* contêm um *link* para um componente do tipo *window* denominado *Counters* (não representado).

Estas funcionalidades pemitem ao utilizador aceder ao mapa em qualquer janela do menu e aceder aos contadores a partir de qualquer janela da aplicação. Estas duas funcionalidades são essenciais, dado que o utilizador recorrerá com frequência ao mapa e aos contadores.

Capítulo 5

Implementação

Este capítulo, tal como no anterior, inicia-se com a apresentação da estrutura geral comum e, de seguida, apresenta pormenores de implementação relativos a cada aplicação.

5.1 Estrutura base do sistema

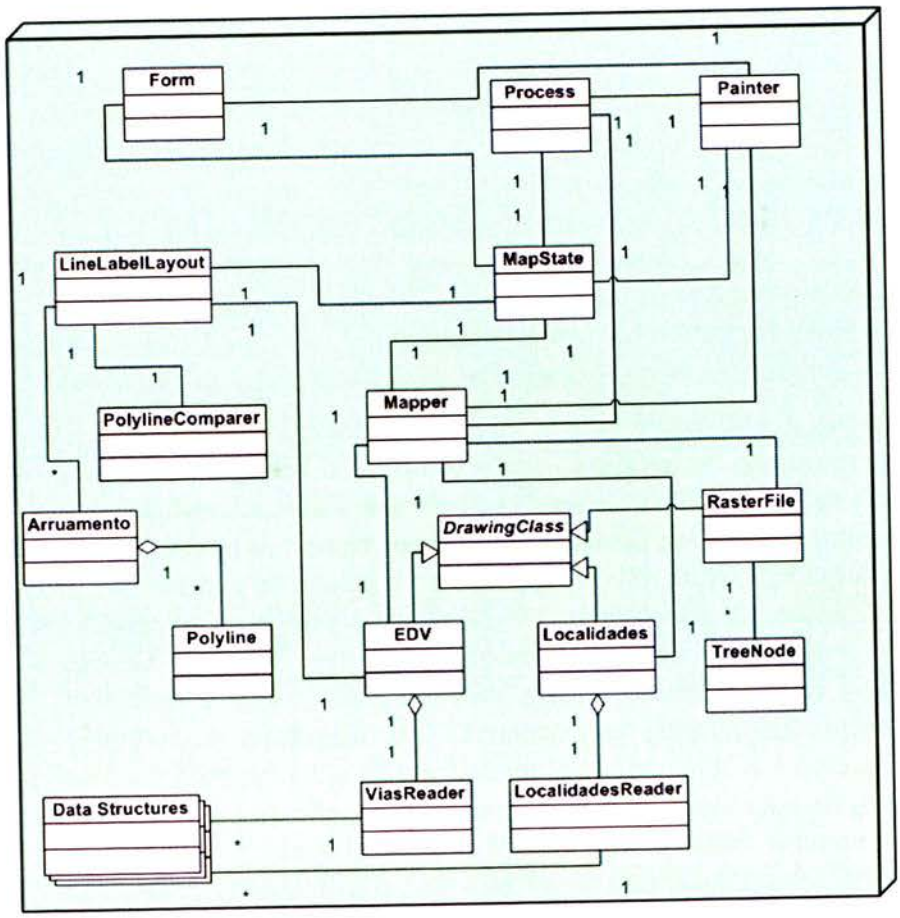

Figura 5.1 – Diagrama de classes comum às duas aplicações

Na figura 5.1, é apresentado o modelo lógico de classes correspondente à estrutura comum partilhada pelas duas aplicações.

Nesta figura estão apenas representadas as classes que desempenham um papel de destaque na aplicação. Embora as classes apresentadas implementem funcionalidades idênticas em ambas as aplicações, existem algumas áreas onde os pormenores de implementação são diferentes. Nas secções seguintes é descrita a implementação das várias classes do sistema e à medida do que for sendo necessário, serão descritos os pormenores de implementação relativos a cada aplicação em particular.

5.1.1 Acesso a dados

Na figura 5.1, o conjunto de classes e estruturas (*struct*) que representa as estruturas de dados (*Data Structures*) onde é guardada a informação vectorial está representado como um conjunto uniforme. A sua função é apenas armazenar a informação associada com a cartografia vectorial: coordenadas de eixos de via, nomes de vias, coordenadas das localidades, coordenadas da grelha das cartas militares (no caso do *MapAdventure Studio*) e polígonos que representam a hidrografia, altimetria e limites administrativos (no caso do *MapAdventure*). Estas estruturas armazenam ainda os respectivos estilos associados a cada tipo de cartografia vectorial.

De modo a fornecer uma melhor compreensão sobre o conceito que suporta a organização interna da cartografia vectorial, segue-se uma breve explicação. A informação vectorial está indexada por camada e, para cada camada, são definidos estilos que permitem definir o modo como a cartografia vectorial é desenhada. Cada camada pode ser associado com um determinado nível de detalhe do mapa. A cada camada associa-se, além disso, um nível de *zoom* extraído da respectiva escala. Deste modo, obtém-se a informação necessária para desenhar o mapa num determinado nível de *zoom*. Por exemplo, no caso das vias, a camada com menor profundidade apenas contém informação relativa às auto-estradas, itenerários principais e respectivos estilos. Por sua vez, a camada seguinte adiciona informação acerca das estradas nacionais. O processo continua até se chegar à última camada. Assim, à medida que a camada se torna mais "profunda" a sua informação é mais detalhada.

As classes *LocalidadesReader* e *ViasReader* têm como função principal aceder aos ficheiros com informação vectorial e colocar a informação lida, nas estruturas de dados. Estas classes podem ser vistas como *parsers*[22] que interpretam a informação vectorial dos ficheiros e a colocam em estruturas de dados apropriadas. Nesta fase, é associado a cada objecto vectorial (via, localidade, carta militar ou polígono) a respectiva camada, são efectuadas operações de transformação de coordenadas e é calculado o respectivo rectângulo envolvente (*bounding box*). O cálculo do rectângulo envolvente permite definir o rectângulo mínimo (área) que envolve um determinado objecto vectorial. Este cálculo é necessário, pois é utilizado por diversas classes, mais concretamente em algoritmos de *clipping*. Esta propriedade é particularmente útil quando é necessário aceder a determinada via, localidade ou carta militar. É utilizada, por exemplo, quando o utilizador pretende visualizar um elemento resultante de uma pesquisa. Neste caso, é apenas necessário aceder ao rectângulo envolvente do objecto vectorial seleccionado para visualizar a via ou localidade seleccionada.

---

[22] Um analisador gramatical (*parser*) consiste numa aplicação que divide a informação, em fracções lógicas mais pequenas, de acordo com um conjunto de regras.

No caso do *MapAdventure Studio* existe ainda uma classe adicional denominada *GridReader*, que permite ler a informação vectorial da grelha militar e colocá-la nas estruturas de dados.

O mecanismo de leitura da cartografia *raster* processa-se de modo diferente. A leitura da cartografia *raster* só é efectuada quando é necessário visualizá-la. Contudo, no ínicio da aplicação, é lida para memória a tabela de indexação relativa à cartografia *raster* através das classes *RasterFile* e *Treenode* de modo a permitir identificar as cartas militares adquiridas pelo utilizador.

Relativamente à aplicação *MapAdventure*, também existe uma classe adicional designada por *PolygonReader* que permite ler a informação vectorial associada com a hidrografia, altimetria e limites administrativos e colocá-la nas respectivas estruturas de dados. No entanto, devido à similaridade destas classes com as descritas, estas não são contempladas nesta secção.

5.1.2 Cartografia Vectorial

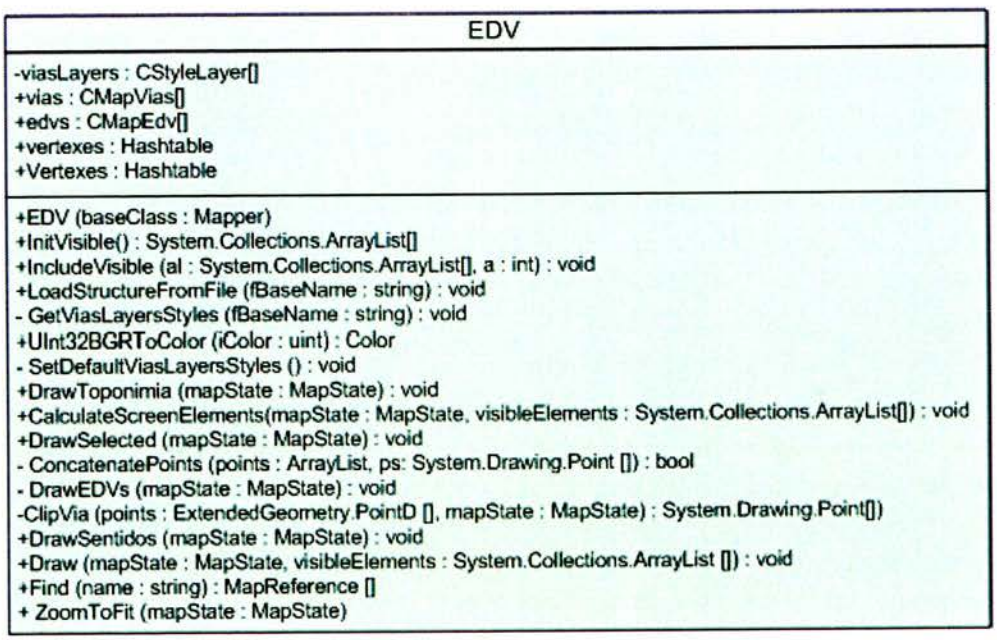

Figura 5.2 – Desenho detalhado da classe *EDV*

As classes *Localidades* e *EDV* acedem às estruturas de dados que as classes discutidas na secção anterior (*LocalidadesReader* e *ViasReader*) preencheram. As classes *Localidades* e *EDV* implementam a classe abstracta[23] *Drawing Class* de modo a tirar partido do polimorfismo.

[23] Uma classe abstracta é uma classe que não pode ser instanciada – ou seja, não pode ter instâncias directas, ou porque a sua descrição é incompleta (tal como métodos em falta para uma ou mais operações) ou porque não foi desenhada com o intuito de ser instanciada, mesmo que a sua descrição seja completa [Rumbaugh et al., 1999]. As classes abstractas são assim abstracções de desenho em que se identificam requisitos a cumprir futuramente, mas que podem ser interpretados (implementados) de formas diferentes. Por outras palavras, ao indicar apenas a sua interface e delegar a implementação, permite-se especializá-la de várias formas.

Estas classes permitem efectuar diversas operações sobre a cartografia vectorial. Entre estas, destacam-se o processo de filtragem da cartografia a visualizar e o respectivo desenho no ecrã. Na figura 5.2, mostra-se o diagrama UML[24] [Rumbaugh et al., 1999] do desenho detalhado da classe física *EDV*. De forma a compreender melhor quais as funções dos métodos desta classe, segue-se uma descrição integral do processo de desenho das vias.

Esta classe é bastante relevante, pois contém alguns métodos essênciais para a comprensão do processo de desenho da cartografia vectorial. Quandoa aplicação se inicia, é invocado o método *LoadStructureFromFile* que permite ler, através da classe *ViasReader*, a informação vectorial a partir de ficheiros externos. Numa fase posterior, ocorre o desenho do mapa que é quando as classe que gere o desenho da catografia do mapa (classe *Mapper*) invoca o método *CalculateScreenElements*. Este permite seleccionar as vias que vão ser visualizadas pelo que, efectua duas funções importantes:

- Pré-selecciona as vias com base na camada a visualizar;
- Filtra novamente as vias através de um algoritmo de *clipping* (*ClipVia*).

O algoritmo de *clipping* efectua o corte das vias através da intercepção da janela de visualização com os eixos das vias. Deste modo, é possível obter apenas os segmentos das vias que necessitam de ser desenhadas, evitando o processamento de vias que não vão ser visualizadas.

Após alguns procedimentos que ocorrem noutras classes, é invocado o método *Draw* que efectua o desenho das vias seleccionadas anteriormente. Este método apenas faz uma chamada aos métodos *DrawEDVs* e *DrawSentidos*: o método *DrawEDVs* pinta as vias seleccionadas segundo o estilo definido e o método *DrawSentidos* contém a lógica necessária para desenhar os sentidos – setas direccionais – nas vias.

No contexto desta classe, importa ainda salientar a funcionalidade do método *ZoomToFit*: este permite a interação entre os elementos de uma pesquisa (fornecidos pelo método *Find*) e o mapa. Assim, quando o utilizador selecciona uma via através dos resultados da pesquisa, a aplicação mostra a localização da via através da invocação deste método que funciona com base nos rectângulos envolventes discutidos anteriomente. Assim, o método apenas acede ao rectângulo envolvente da via seleccionada. Por sua vez, a classe *DrawToponimia* suporta a lógica necessária para o desenho da toponímia. A implementação do desenho da toponímia será discutido em secções posteriores, pois merece uma atenção especial.

Importa ainda referir que as classes que manipulam a cartografia vectorial (*EDV* e *Localidades*) ofereceram alguma "resistência" aquando da portabilidade para a plataforma *.NET Compact Framework*. Esta resistência revelou-se, sobretudo, nas funções gráficas, onde a plataforma *.NETCF* tem maiores limitações relativamente à versão *desktop*. Consequentemente, foram necessárias algumas modificações nos algoritmos de desenho da cartografia vectorial de forma a implementar as funcionalidades existentes na versão *desktop*. Embora estas alterações já estivessem previstas, exigiram algum esforço adicional relativamente ao planeado, resultando num pequeno atraso relativamente ao planeamento inicial.

[24] *Unified Modeling Language*

Tal como na secção anterior, também aqui as duas aplicações possuem algumas classes adicionais que importa destacar. Além das classes apresentadas, a aplicação *MapAdventure Studio* possui uma classe denominada *Grid* que efectua funções semelhantes à classe *EDV* para os elementos vectoriais da grelha. Por outro lado, a aplicação *MapAdventure* possui uma classe adicional, designada por *Polygon*, que gere a cartografia vectorial referente aos polígonos que representam a hidrografia, altimetria e os limites administrativos.

As restantes classes que manipulam cartografia vectorial não serão contempladas, pois apresentam uma estrutura e funcionalidade semelhante à descrita para a classe *EDV*.

5.1.3 Mapa

As classes *MapState* e *Mapper* desempenham um papel central nas duas aplicações. Estas classes permitem, respectivamente, o controlo do estado do mapa e a coordenação das operações de desenho de toda a cartografia. Na figura 5.3[25], é representado a composição das duas classes.

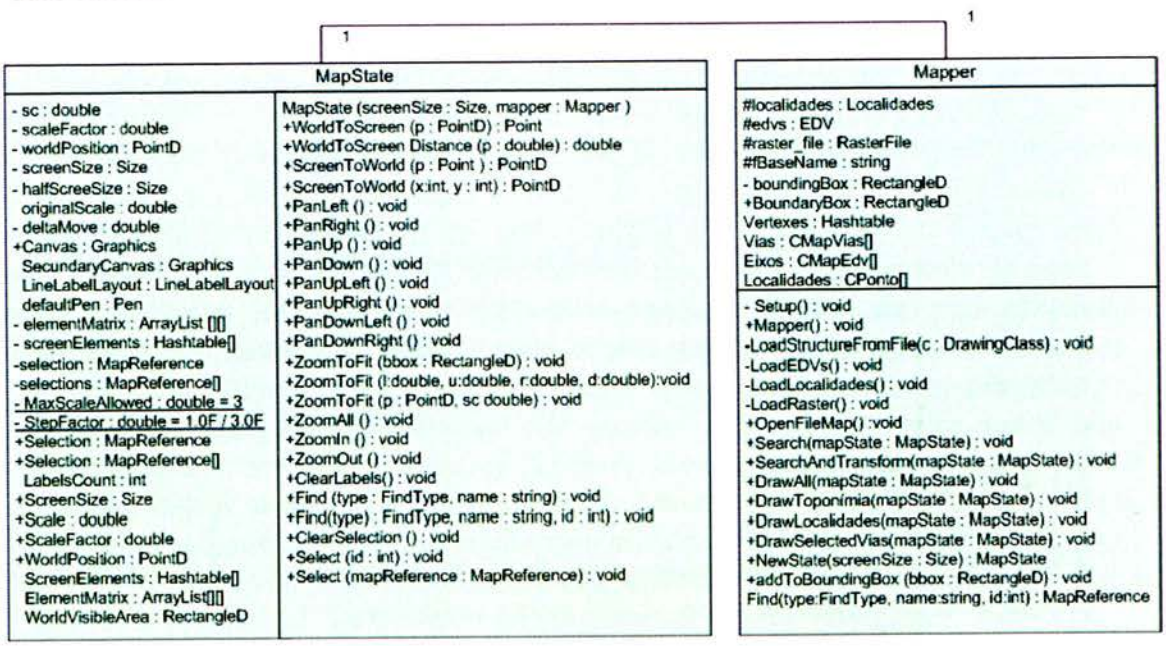

Figura 5.3 – Desenho detalhado das classes *Mapper* e *MapState*

A classe *MapState* possui duas funções principais: guarda os vários dados relativos ao estado do mapa e permite realizar diversas operações de navegação (*zoom in/out, pan up/down/left/right e zoom to fit*) sobre o mapa. A informação relativa ao estado do mapa contempla a escala actual, a área visível (*WorldVisibleArea*) e as coordenadas do centro do mapa (*worldPosition*). Esta informação é manipulada pelos diversos métodos da classe que efectuam as operações de navegação. Assim, cada vez que o utilizador navega no mapa, as classes que implementam a interface gráfica invocam os métodos de navegação desta classe e a informação do estado do mapa é actualizada de modo a reflectir a operação realizada.

[25] Nesta figura, os métodos da classe *MapState* foram colocados horizontalmente devido ao excessivo espaço vertical ocupado pela classe.

Posteriormente, esta informação é utilizada na fase de desenho, pelas classes que manipulam a cartografia, de modo seleccionar a cartografia vectorial e *raster* a ser visualizada. Esta classe disponibiliza ainda métodos que permitem a pesquisa de todos os elementos da cartografia vectorial. Estes apenas invocam os métodos de pesquisa das classes que manipulam a informação vectorial. Assim, esta classe funciona como elemento centralizador da informação geográfica relativa ao mapa. Esta centralização acentua a flexibilidade da aplicação, pois permite que outros tipos de cartografia (pontos de interesse, *tracks*, etc) possam ser adicionados sem grande impacto na estrutura da aplicação.

A classe *Mapper* efectua a gestão da leitura dos dados dos ficheiros e das operações de desenho da cartografia. Os métodos que permitem efectuar a leitura da informação cartográfica recorrem ao uso de *threads* para optimizar os mecanismos de acesso e leitura de ficheiros. Deste modo, é atribuída a cada *thread* a responsabilidade de ler um tipo de cartografia. Através do polimorfismo, são invocados, por cada *thread*, os métodos *LoadStructureFromFile* das classes que manipulam a cartografia. Esta classe possui ainda os métodos que gerem o desenho do mapa. Estes apenas invocam os métodos de desenho (*Draw*) das classes que gerem a cartografia. No entanto, há que ter em conta a ordem de desenho da cartografia que é gerida por esta classe.

Em suma, estas classes representam uma camada de abstracção sobre o mapa e sobre as suas funcionalidades, o que torna o manuseamento da informação relativa ao mapa mais simples e flexível. Deste modo, é possível controlar todo o comportamento do mapa, a partir destas classes, sem ter conhecimento das camadas de implementação inferiores (acesso a dados, manipulação da cartografia, etc), o que permite que o futuro desenvolvimento da aplicação possa ser efectuado sem a necessidade de um conhecimento profundo das restantes classes do sistema. Esta característica reveste-se de uma importância acrescida, pois confere um grande nível de portabilidade ao código desenvolvido e facilita o posterior desenvolvimento das aplicações.

5.1.4 Toponímia

A toponímia é uma área de grande relevo nesta aplicação, devido à sua importância para a orientação do utilizador no mapa. A abordagem deste problema exige uma análise do modo como a toponímia é colocada no mapa. Neste caso, a pergunta que se coloca é: como colocar a toponímia nas vias de modo a ser exibida sem ambiguidades e permitir uma compreensão rápida e precisa da informação? A resposta a esta pergunta será dada, no parágrafo seguinte, à medida que forem descritas as funcionalidades das classes associadas com esta temática. Antes de proceder à descrição das classes importa ainda esclarecer alguns conceitos: neste contexto, um arruamento (*Arruamentos*) representa uma via e é constituido por várias polilinhas (*Polyline*). Por sua vez, cada polilinha corresponde a um conjunto de segmentos de recta em sequência. De seguida, descreve-se o processo de colocação da toponímia nas vias.

O processo de colocação da toponímia baseia-se na escolha, para cada arruamento, da polilinha mais adequada para a colocação da toponímia. A escolha da polilinha é realizada segundo critérios que permitem seleccionar a polilinha que tem melhores condições para asseguar a visualização correcta da informação toponímica. Neste caso optou-se por dois tipos de colocação da toponímia: no interior das vias e junto às vias através de etiquetas. As etiquetas são usadas para distinguir as vias principais (auto-estradas e itenerários principais). Nas restantes vias a toponímia é colocada no interior da via. No entanto, esta opção é

condicionada pelo espaço disponível no interior da via que, por sua vez, depende do nível de detalhe do mapa. Assim, quando os níveis de *zoom* não são muito profundos, apenas estão assinaladas as estradas principais.

O processo inicia-se com o manipulação da informação vectorial das vias por parte das classes *Arruamento* e *Polyline*. Estas processam a informação vectorial das vias de modo a permitir a manipulação geométrica das polilinhas e arruamentos por parte da classe *LineLabelLayout*. Posteriormente, a classe *LineLabelLayout* utiliza a informação obtida dos vários arruamentos e selecciona a melhor polilinha para cada arruamento. O método utilizado para a selecção das polilinhas depende do método de colocação da toponímia. No caso da toponímia ser colocada no interior da vias é utilizada um comparador de polilinhas – *PolylineComparer*. Esta classe funciona como um filtro das várias polilinhas, ordenando-as por ordem de preferência pelo que se constitui de uma série de funções heurísticas que permitem ordenar as várias polilinhas usando vários critérios predefinidos. A selecção tem como base alguns critérios relativos à geometria das polilinhas calculados anteriormente pela classe *Polyline*: distância total, ângulo máximo com a horizontal, ângulo máximo entre segmentos justapostos e número de pontos que compõe a polilinha. Deste modo, pretende-se evitar que a toponímia seja colocada na via com demasiada inclinação, que fique fora dos limites da via ou que sofra quebras abruptas de direcção. Estes critérios são essenciais, pois permitem obter a melhor localização para a toponímia tendo em conta a sua legibilidade. Por outro lado, se a toponímia for colocada em etiquetas, apenas se selecciona a polilinha central do arruamento. Depois da selecção das polilinhas, a classe *LineLabelLayout,* verifica se existem sobreposições de toponímia. Caso existam, retira a toponímia que referencia a via menos importante e desenha a toponímia no mapa.

Esta solução é ideal para a aplicação *MapAdventure Studio*. No entanto, revelou-se insustentável na aplicação *MapAdventure* devido às elevadas exigências em termos de processamento. Depois de efectuados algumas medições de desempenho, verificou-se que os métodos de selecção de polilinhas são aqueles que exigem maior capacidade de processamento. Uma solução possível que poderá ser implementada passa por colocar a toponímia usando apenas etiquetas. Deste modo, é possível contornar o cálculo exaustivo da selecção de polilinhas, pois apenas se efectua o cálculo da polilinha média do arruamento. No entanto, o baixo desempenho na colocação das polilinhas também se deve à baixa eficiência dos métodos de desenho. Este assunto será explorado no capítulo seguinte.

5.1.5 Visualização

Um assunto pertinente que surge em todos os sistemas que exigem elevado processamento ao nível gráfico é a optimização. No caso particular deste sistema existe uma classe – *Painter* – que dedicada exclusivamente a optimizações efectuadas a este nível. Esta classe implementa métodos que permitem optimizar o desempenho da aplicação em termos gráficos. Nesta classe é utilizada uma técnica bastante conhecida de optimização gráfica – *double buffer* – e é implementada outra técnica que permite efectuar o desenho do mapa através de camadas – não confundir as camadas aqui referidas com as camadas referidas anteriormente e que determinam o nível de detalhe de cada nível de *zoom*. Neste caso, as camadas referem-se a *bitmaps* utilizados para desenhar do mapa em memória.

O *double buffer* é uma técnica que consiste em criar o mapa em memória antes de este ser mostrado ao utilizador. Neste mapa são realizadas todas as operações de desenho como se

estivessemos a desenhar para o ecrã. Posteriormente, quando o mapa é mostrado no ecrã apenas se efectua uma cópia do mapa em memória para o ecrã. Deste modo, previne-se o utilizador de visualizar as versões intermédias e incompletas do mapa (*flicker*) e a aplicação aparenta ser mais rápida, o que cria na percepção do utilizador a imagem de uma aplicação mais robusta.

A segunda técnica contempla o desenho do mapa por camadas. Esta técnica possibilita a paralelização do desenho do mapa num ambiente *multi-threaded*: cada *thread* é responsável pelo desenho de uma camada. Neste sistema, foram utilizadas apenas duas camadas. Na aplicação *MapAdventure Studio,* na primeira camada, são desenhadas a cartografia *raster*, as vias e a grelha representativa das cartas militares. Na segunda camada, são desenhadas a toponímia e as localidades. Este processo termina com a junção – *blending* – das duas camadas antes do mapa ser copiado para o ecrã (*double buffer*).

De modo a permitir uma melhor compreensão dos processos utilizados para gerar o mapa, a figura 5.4 apresenta o *flow* de interacções desde que a interface gráfica efectua o pedido à classe *Painter* para desenhar o mapa, até que este é efectivamente desenhado.

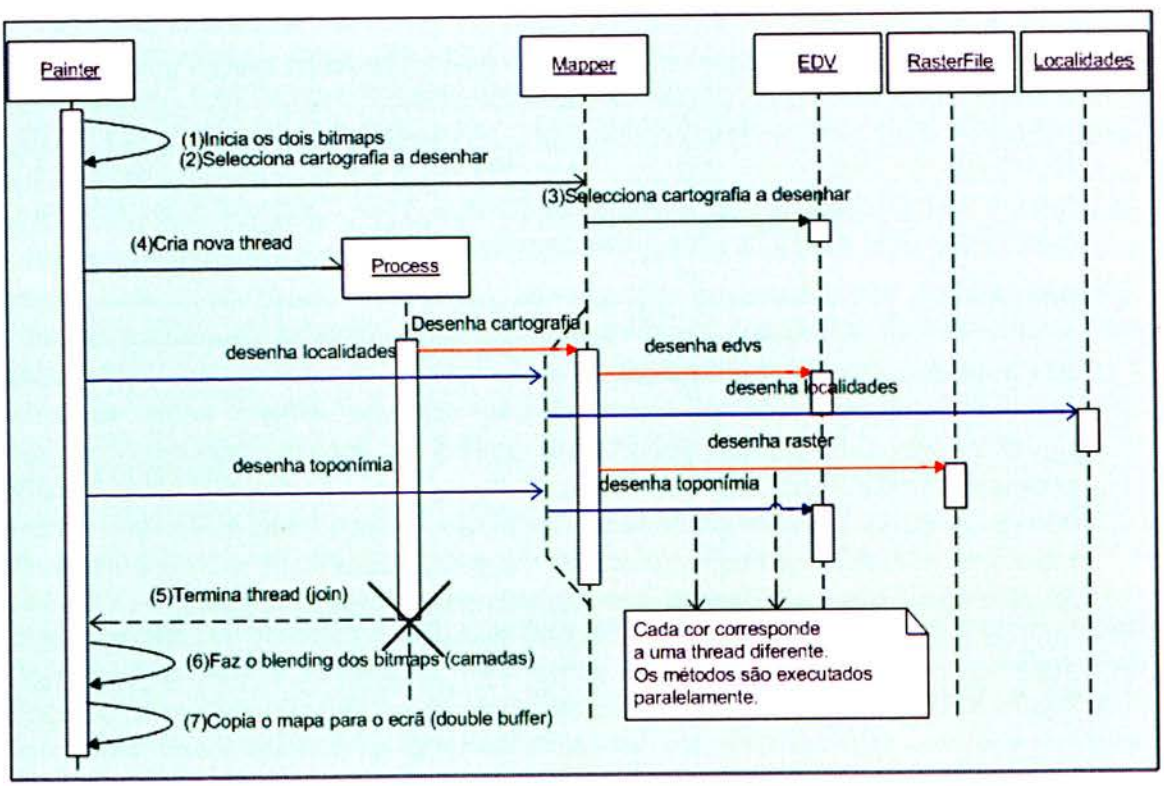

Figura 5.4 – Diagrama de sequência do processo de desenho do mapa

Na aplicação *MapAdventure* esta técnica do desenho por camadas não foi implementada, pois não existe a possibiliade de aplicação ser executada num ambiente multi-processador como na plataforma *desktop*. Por outro lado, a criação de *threads* apenas iria criar um *overhead* adicional no processo de desenho do mapa diminuindo o desempenho da aplicação.

Verificou-se, entretanto, que as optimizações efectuadas a este nível não são suficientes para a aplicação *MapAdventure*, visto que o respectivo *hardware* esta apresenta um desempenho bastante inferior ao pretendido, sobretudo no desenho do mapa. Assim sendo, são necessárias

soluções alternativas que permitam solucionar este problema. No capítulo seguinte serão analisadas com maior detalhe algumas soluções propostas pelo autor.

5.1.6 Exportação de Regiões

A aplicação *MapAdventure Studio* disponibiliza uma interface de comunicação com a aplicação *MapAdventure* que permite ao utilizador efectuar a exportação das regiões seleccionadas. Assim, a aplicação *MapAdventure Studio* contém uma classe denominada *RegionDetails* especialmente dedicada à gestão da informação associada com as regiões seleccionadas. Esta classe está intimamente ligada à interface gráfica, já que os dados das regiões seleccionadas pelo utilizador, são direccionados directamente da interface gráfica para esta classe. Assim, cada vez que o utilizador adiciona ou elimina uma determinada região, a classe *RegionDetails* é actualizada. A classe contém a seguinte informação para cada região: tamanho total da região, cor associada à região (legenda), coordenadas da região e cartas militares abrangidas pela região. Quando o mapa é desenhado, as classes que compõem a interface gráfica acedem à informação contida na classe *RegionDetails* e processam o desenho das regiões seleccionadas.

Quando o utilizador exporta as regiões seleccionadas, a classe *RegionDetails* passa a informação das regiões para a classe *RasterFile*. Por sua vez, esta classe cria uma tabela de indexação idêntica à utilizada para a leitura dos ficheiros *raster* adquiridos pelo utilizador. Contudo, antes das imagens serem efectivamente exportadas, estas classe efectua a sua encriptação através do algoritmo de encriptação simétrica (mesma chave para encriptar e desencriptar as mensagens) – *Blowfish*[26]. Posteriormente, quando se processa a leitura da informação *raster* no *Pocket PC*, o *MapAdventure* desencripta a informação e lê os ficheiros *raster* do mesmo modo que a aplicação *MapAdventure Studio*.

5.1.7 Algoritmo de *Brushing*

A necessidade de desenvolver um algoritmo de *brushing*[27], para as vias da aplicação *MapAdventure*, surgiu devido ao facto da plataforma *.NETCF* não disponibilizar métodos que permitam desenhar linhas (vias) com largura variável. Esta funcionalidade é necessária para diferenciar os vários tipos de vias que são representadas no mapa – as vias mais importantes são representadas com uma largura maior do que as vias com menor importância. Assim, a largura das vias é directamente proporcional à sua importância. Esta funcionalidade permite ao utilizador identificar rapidamente as principais vias e, de um modo instintivo, situar-se no mapa.

De modo a solucionar este problema, foi desenvolvido um algoritmo que permite criar linhas com largura variável utilizando duas figuras geométricas (círculo e rectângulo).

Na figura 5.5 está representada uma via através de uma polilinha. Como se pode ver na figura, a polilinha (representada a linha espessa) é composta por várias segmentos de recta que, neste

[26]Este algoritmo utiliza técnicas de cifragem em bloco e permite que as chaves possam ter qualquer comprimento (128 *bits* é o mais corrente na actualidade).

[27]Este tipo de algoritmo é normalmente associado ao desenho de linhas (ou formas geométricas) com espessura variável.

contexto, são designados por eixos de via. Os pontos de ligação entre os eixos de via são designados por nós. Interessa ainda referir que uma via pode ser composta por várias polilinhas.

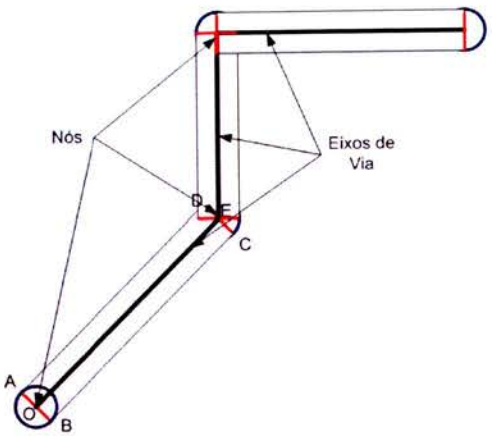

Figura 5.5 – Via representada através de uma polilinha.

O algoritmo de *brushing* processa cada eixo de via separadamente. Através das coordenadas dos nós – guardadas nas estruturas de dados referidas anteriormente – de cada eixo de via (pontos O e E), são calculados os pontos A, B, C e D. O cálculo destes pontos baseia-se na largura pretendida da via e na inclinação do eixo de via.

Para calcular estes pontos é necessário calcular a inclinação (declive) do eixo de via (através dos pontos O e E). Após termos calculado esta inclinação, podemos determinar através de métodos matemáticos a inclinação da recta que intercepta os pontos A e B e C e D. Posteriormente, podemos calcular os pontos A, B, C e D. Este cálculo é efectuado através de um algoritmo proprietário da *ParadigmaXis* que utiliza a largura da via e a inclinação calculada anteriormente para determinar estes pontos. Posteriormente, é desenhado um rectângulo, que tem como vértices os pontos A, B, C e D, e um círculo com centro no nó inicial (ponto O) e com um raio de comprimento igual a metade da largura pretendida para a via. Apenas no último eixo de via são desenhados dois círculos (nos extremos do eixo de via). O desenho dos círculos visa suavizar as mudanças de direcção que existem ao longo de uma via, de modo a tornar o desenho das vias mais fluido. Após processado este eixo de via, os restantes são processados do mesmo modo. A únicas excepções abrangem os eixos verticais e horizontais, pois neste tipo de eixos as coordenadas dos vértices do rectângulo podem ser calculadas directamente, não precisando de ser processados pelo algoritmo descrito. Deste modo, o *overhead* associado a este algoritmo é evitado neste tipo de vias.

Uma solução óbvia para este problema poderia basear-se no desenho de linhas paralelas juntas ao eixo de via até se atingir a largura de via pretendida. No entanto, esta solução criaria demasiado *overhead* devido às operações de desenho das várias linhas. Na solução adoptada, esta operação é feita apenas em dois passos:

- **Cálculo dos vértices do rectângulo:** Este cálculo foi bastante optimizado, pelo que o *overhead* criado não é significativo.

- **Desenho do rectângulo:** Esta operação consiste apenas num passo simples e usa uma função disponibilizada pelo sistema para desenhar o rectângulo.

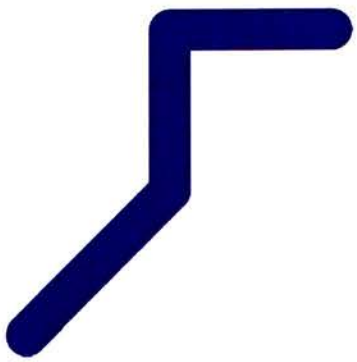

Figura 5.6 – Via processada pelo algoritmo de *brushing*.

Na figura 5.6, podemos verificar o resultado final do algoritmo quando os círculos e os rectângulos representados na figura 5.5 são preenchidos com a cor azul.

5.1.8 *GPS* e *Tracks*

A implementação destas funcionalidades decorre no momento de escrita deste relatório. Contudo, será feita uma breve descrição das classes que implementam estas funcionalidades, de modo a fornecer ao leitor uma ideia acerca do desenvolvimento nestas áreas.

No caso do *GPS* [James, 2000], recorreu-se a uma biblioteca de código aberto[28] (*open source*) bastante reconhecida na área do desenvolvimento para dispositivos móveis, para implementar as diversas funcionalidades relativas ao *GPS*. Esta biblioteca também diponibiliza métodos eficientes de acesso às portas série do dispositivo móvel, o que permitiu um desenvolvimento mais rápido e linear do que o inicialmente previsto para esta fase. Consequentemente, foram apenas efectuadas alterações às classes relativas ao *GPS*, dado que a ligação do dispositivo móvel ao dispositivo *GPS* é totalmente gerida pelas classes da biblioteca. A classe *GPS* é totalmente responsável pela implementação das funcionalidades *GPS*. Estas é constituída por um *parser* que interpreta a informação do dispositivo *GPS* e por várias funções adicionais que permitem "disparar" eventos quando ocorrem factos relevantes: por exemplo, quando ocorre uma mudança no nível de sinal recebido ou quando o utilizador se desloca. À medida que o utilizador se desloca, a classe *GPS* "fornece" à classe *MapState* informação acerca das coordenadas do utilizador, usando os eventos da classe *GPS*. Deste modo, a classe *MapState* actualiza o estado do mapa, permitindo a sincronização em tempo real, do mapa com a posição do utilizador. Do mesmo modo, a interface gráfica também implementa os eventos da classe *GPS*, de modo a actualizar a informação relativa ao estado do *GPS*.

A classe *Track* permite gerir a informação relativa aos *tracks*. Esta classe possui métodos que permitem armazenar e ler *tracks* a partir de um ficheiro. Os ficheiros de *tracks* são

[28] A biblioteca utilizada é designada por *OpenNETCF.org Smart Device Framework* [Opennetcf, 2005] e surgiu com o objectivo de colmatar as limitações da *.NETCF*.

constituídos por dados sobre as coordenadas dos vários *waypoints* que compõem as *tracks*. Assim, quando o utilizador pretende guardar uma *track*, esta classe apenas regista os *waypoints* criados pelo utilizador no ficheiro apropriado. Por outro lado, quando o utilizador pretende ler uma *track*, esta classe apenas lê as coordenadas dos *waypoints* e desenha a *track* no mapa usando métodos de desenho similares aos utilizados na cartografia vectorial. Deste modo, cada vez que o mapa é desenhado, o método *Draw* é invocado e as *tracks* são desenhadas. Posteriormente, esta classe poderá ser extendida de modo a fornecer métodos que permitam a manipulação de rotas e a implementação de funcionalidades de navegação adicionais.

5.2 Resultados

Esta secção pretende resumir os resultados obtidos ao longo do decorrer do projecto.

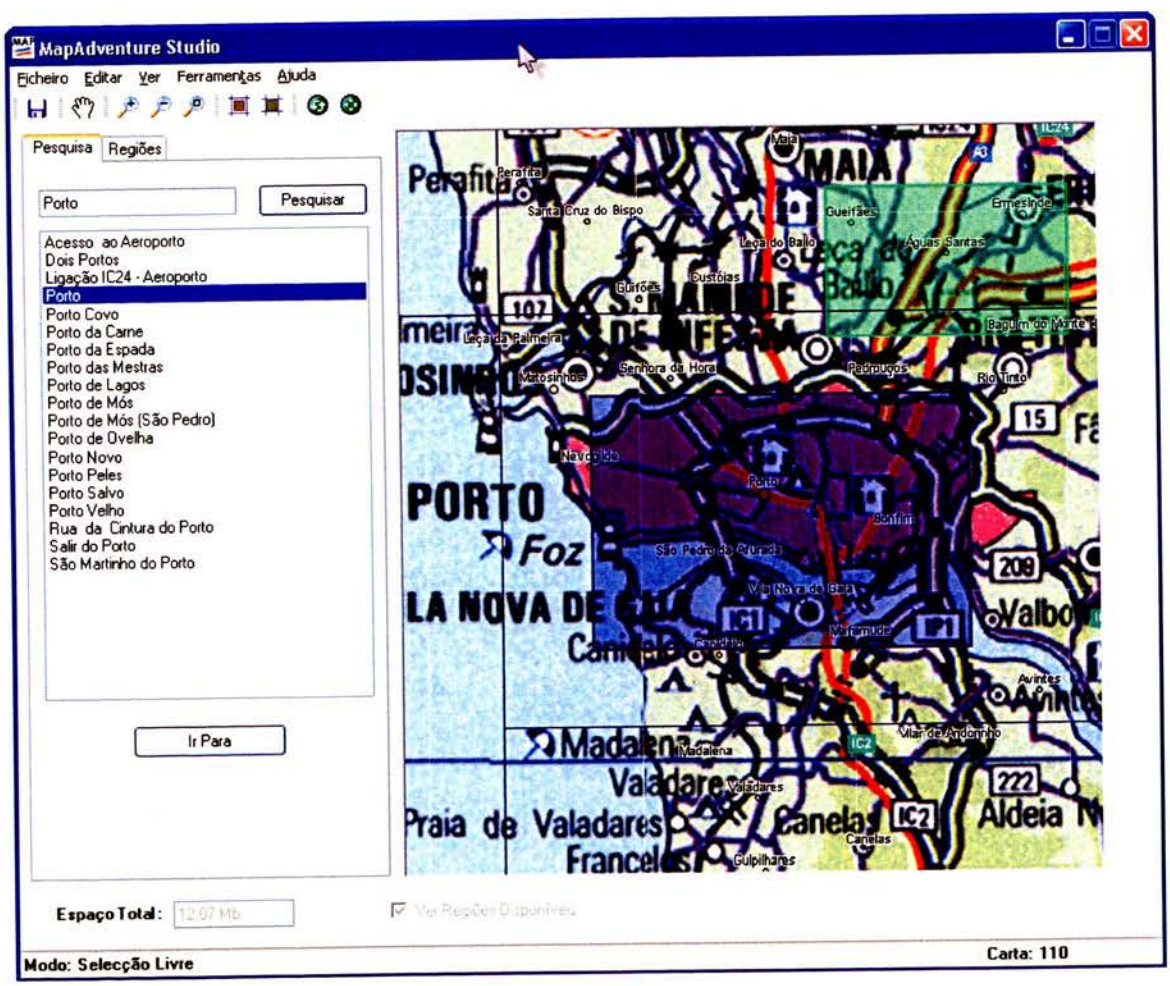

Figura 5.7 – Screenshot da aplicação *MapAdventure*

Como já foi referido, a aplicação *MapAdventure Studio* foi totalmente implementada, restando apenas implementar o processo de validação das cartas militares adquiridas pelo utilizador. Toda a interface gráfica foi implementada segundo os requisitos especificados inicialmente. Segundo os testes efectuados por alguns utilizadores, a aplicação disponibiliza funcionalidades intuitivas e apresenta uma curva de aprendizagem bastante rápida. Nesta

aplicação, pretendia-se que o utilizador aprendesse de um modo rápido a utilizar as principais funcionalidades da aplicação. Este facto conduziu, como se pode ver na figura 5.7, a que as funcionalidades estivessem expostas ao utilizador através de barras de ferramentas flutuantes e não "escondidas" em menus, de modo a facilitar a usabilidade da aplicação. A área referente às pesquisas e às regiões está sempre apenas à distância de um clique o que permite ao utilizador rapidamente alternar entre as duas áreas e obter informação sobre as regiões seleccionadas.

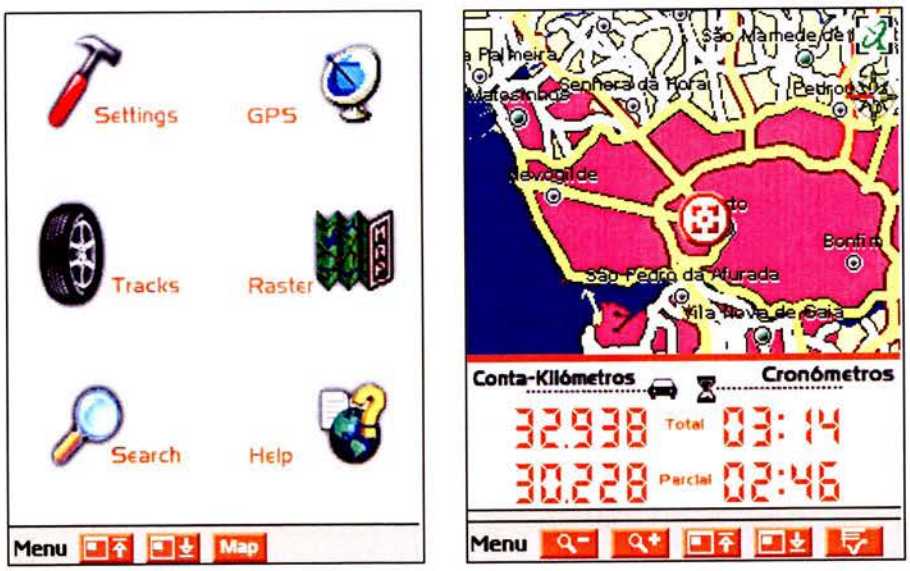

Figura 5.8 – Screenshots da aplicação *MapAdventure*

A aplicação *MapAdventure* apenas foi utilizada como prova de conceito de modo a testar as capacidades da plataforma *.NET Compact Framework*. Como já foi referido atrás, o desempenho da aplicação ficou aquém do esperado. Embora não se tenham encontradas soluções para o problema, são sugeridas possíveis soluções no próximo capítulo. No entanto, o desenvolvimento da aplicação foi construtivo, pois permitiu conhecer as particularidades desta plataforma e as limitações quando se desenvolvem aplicações para dispositivos móveis. Entre uma das vantagens encontradas no desenvolvimento da aplicação *MapAdventure* destaca-se a existência de várias *API*'s que facilitam o desenvolvimento de vários tipos de aplicações e a facilidade encontrada no desenvolvimento da interface gráfica devido às ferramentas disonibilizadas pelo *VS .NET*. Como se pode verificar na figura 5.8, todos os ícones foram projectados de modo a permitirem uma fácil interpretação e com uma dimensão que permite a interacção através do toque do dedo. Os botões (cor laranja) na área inferior permitem alternar rapidamente entre o menu e o mapa.

Capítulo 6

Conclusões

Neste capítulo é primeiramente abordado uma pequena análise de resultados, seguindo-se um conjunto de funcionalidades ou optimizações que poderão ser realizadas sobre o trabalho desenvolvido. No final, é apresentado um pequeno resumo e conclusões finais referentes a todo o trabalho.

6.1 Análise de Resultados

6.1.1 *MapAdventure Studio*

No momento de escrita deste relatório, a aplicação *MapAdventure Studio* está prestes a entrar na fase de comercialização, encontrando-se, actualmente, numa fase de controlo de qualidade e depuração por parte da *ParadigmaXis*.

Como aproximação à metodologia de *eXtreme Programming*, a aplicação foi sujeita a testes de usabilidade ao longo de todo o projecto. Este basearam-se na utilização da aplicação por parte de alguns utilizadores externos à empresa que, depois de terem testado a aplicação, reportavam as suas observação ao estagiário. Estes testes revelaram-se bastante eficazes, pois permitiram identificar alguns pormenores da interface gráfica que necessitaram de ser alterados de modo a permitir a sua plena usabilidade e ajudaran na identificação de alguns erros de programação que foram prontamente resolvidos.

Actualmente, novas funcionalidades têm também vindo a ser sugeridas, as quais permitem prever a viabilidade do desenvolvimento de uma nova versão. Em suma, a aplicação tem vindo a revelar-se robusta e bem acolhida.

6.1.2 *MapAdventure*

No momento de escrita deste relatório, a aplicação *MapAdventure* ainda se encontra em fase de desenvolvimento, nomeadamente nas áreas que englobam as *tracks* e as funcionalidades relativas ao *GPS*. No entanto, nesta fase não se prevêm problemas significativos de

imlementação, já que o desenvolvimento destas funcionalidades se tem revelado bastante linear e tem decorrido sem grandes sobressaltos. Ao longo do desenvolvimento do projecto surgiram alguns problemas de portabilidade, essencialmente na área relativa aos algoritmos de desenho das classes que manipulam a cartografia. Neste caso, foram necessárias algumas alterações das funções gráficas, devido à limitação da *.NETCF* nesta área. No entanto, a alteração do código relativo às restantes classes apenas envolveu modificações minímas, de modo a tornar-se compatível com a plataforma *.NETFC*. Este facto leva o autor a concluir que, de um modo geral, a portabilidade entre as duas plataformas tem uma exequibilidade aceitável.

Contudo, a plataforma *.NETCF* revelou-se incapaz de satisfazer totalmente os requisitos gráficos que foram estipulados inicialmente para esta aplicação. Esta incapacidade revelou-se, sobretudo, nos elevados tempos de desenho do mapa. Este facto deve-se principalmente à baixa eficiência das funções gráficas na plataforma *.NETCF*. Assim, ficou provado que a utilização desta pataforma para o desenvolvimento de aplicações exigentes, em termos gráficos, não é viável. Deste modo, são propostas, nas secções seguintes, algumas soluções que permitem optimizar o desempenho da aplicação.

Neste contexto, importa também salientar a grande vantagem associada ao desenvolvimento de código conseguida através da utilização da aplicação *Microsoft Visual Studio .NET*. Esta vantagem foi particularmente relevante aquando do desenvolvimento da interface gráfica. Mais concretamente, na utilização das ferramentas de depuração e na utilização do emulador. Estas características possibilitaram a rápida identificação de erros de programação e permitiram que os testes efectuados à aplicação *MapAdventure* fossem realizados de um modo bastante rápido e eficiente, tornando o ciclo de desenvolvimento mais rápido.

6.2 Trabalho Futuro

É apresentado de seguida um conjunto de funcionalidades e de optimizações propostas pelo autor para inclusão futura em ambas as aplicações. Relativamente à aplicação *MapAdventure Studio*, é proposta a adição de novas funcionalidades que visam aumentar o número de ferramentas disponibilizadas ao utilizador na área da navegação. Na secção dedicada à aplicação *MapAdventure* são apresentadas algumas sugestões de optimização que visam melhorar o desempenho no domínio relativo ao desenho do mapa.

6.2.1 *Map Adventure Studio*

Embora a funcionalidade actual desta aplicação seja apenas o suporte à aplicação *MapAdventure*, a sua gama de funcionalidades poderia ser extendida de modo a suportar ferramentas de navegação adicionais. Assim, poderiam ser implementadas funcionalidades que possibilitassem a criação, edição e exportação de rotas, de forma a permitir a sua utilização na aplicação *MapAdventure*. Adicionalmente, poder-se-ia implementar um módulo *GPS* que permitisse obter as funcionalidades actualmente existentes na aplicação *MapAdventure*. Deste modo, aumentava-se o mercado-alvo desta aplicação, abrangendo os utilizadores que não possuem dispositivos móveis, mas apenas computadores portáteis.

6.2.2 *MapAdventure*

Como já foi referido, a plataforma *.NETCF* não apresenta as condições necessárias para aplicações que exijam um elevado desempenho gráfico. Assim, a solução proposta pelo autor passa pelo desenvolvimento de uma aplicação híbrida, isto é, uma aplicação composta por *managed code* e *unmanaged code*. A solução contempla o desenvolvimento de uma biblioteca (*.DLL*), em *unmanaged code* (através da linguagem *C++*), que implementa as funções críticas relativas ao desenho do mapa e o desenvolvimento, em *managed code*, das restantes funcionalidades da aplicação através da plataforma *.NETCF*, usando a linguagem *C#*. O código *unmanaged* seria embebido no código *managed* através do mecanismo *P/Invoke*.

Existem vários tipos de interoperação entre *unmanaged code* e *managed code*. No entanto, apenas será analisado o caso unidireccional em que código *unmanaged* é acedido por código *managed* através do mecanismo *Platform Invoke*. Os restantes casos não serão considerados, pois envolvem a manipulação de objectos *COM* para efectuar a comunicação entre os dois contextos, o que adiciona um *overhead* considerável, inviabilizando a sua implementação.

Através da solução híbrida, é possível tirar partido das vantagens oferecidas pelos dois tipos de desenvolvimento: Por um lado, o desenvolvimento através da linguagem C# proporciona um ambiente de desenvolvimento rápido onde a interface gráfica pode ser rapidamente implementada. Por outro lado, a utilização de *unmanaged code* permite o acesso a um ambiente de maior desempenho, onde as secções críticas da aplicação – manipulação de cartografia vectorial e desenho do mapa – podem ser implementadas. De modo a prever o aumento de desempenho ganho através da utilização de *unmanaged code* foram efectuados alguns testes. Estes testes permitem comparar o desempenho de funções de desenho em ambientes exclusivamente *managed* face a ambientes onde o código *unmanaged* está embebido em código *managed*.

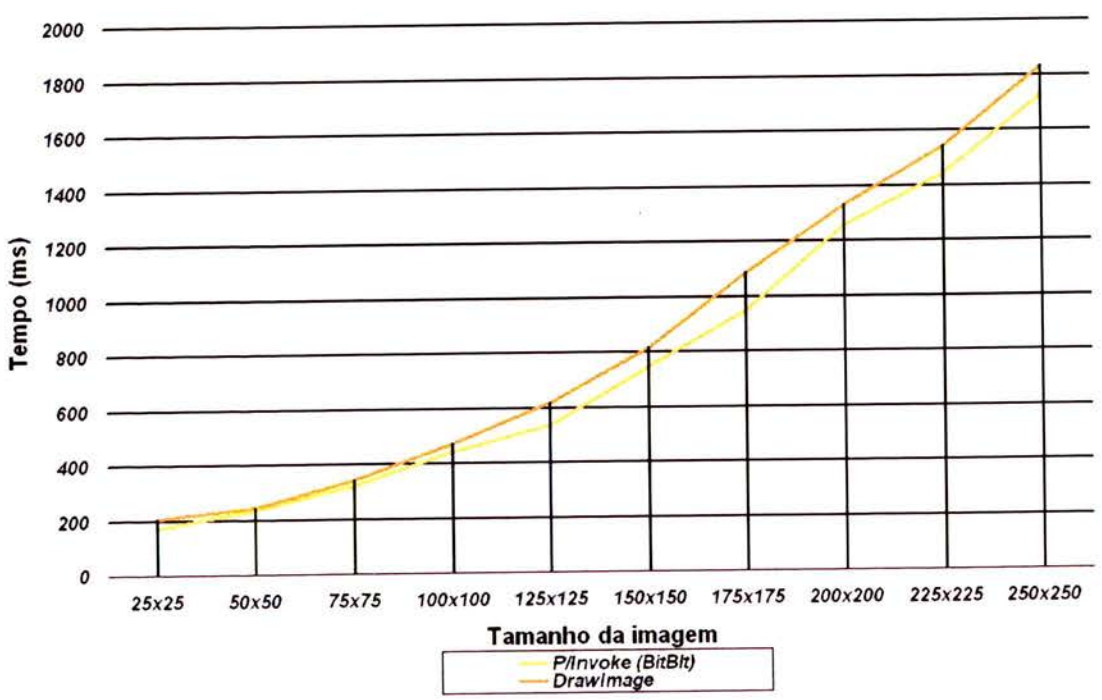

Figura 6.1 – Desempenho das funções gráficas *DrawImage* e *BitBlt* (com o mecanismo *P/Invoke*).

Na figura 6.1 estão representados os resultados dos testes que permitem comparar o desempenho de funções gráficas nos dois ambientes distintos: os valores representados a laranja indicam o tempo de desenho, em milisegundos, obtidos em ambientes exclusivamente *managed*. Por sua vez, os valores representados a amarelo indicam o desempenho obtido em ambientes que *unmanaged code* está embebido em *managed code*.

Os testes foram efectuados sobre duas funções críticas usadas na computação gráfica e que são consideradas como *benchmarks* na análise de desempenho nesta área: *Bitblt* (ambiente *unmanaged*) e *DrawImage* (ambiente *managed*). Estas permitem efectuar a cópia de um bloco de *pixels* de uma posição de memória para o ecrã [Chandler et al., 2001]. As duas funções testadas são bastante similares, o que se deve ao facto da função *DrawImage* usar a função *BitBlt* na sua implementação interna. No entanto, a função *DrawImage* disponibiliza funcionalidades adicionais, tais como o desenho parcial da imagem, a alteração da escala da imagem, entre outras.

Como se pode ver pelos resultados obtidos, pode-se comprovar que a função *BitBlt* possui um desempenho superior à respectiva versão em *managed code* (*DrawImage*). Esta diferença de desempenho deve-se principalmente ao facto da cada chamada à função *DrawImage* ser interpolada pelo *CLR*, o que adiciona um *overhead* adicional em cada invocação. No mesmo sentido, verificou-se que após várias chamadas consecutivas à função *DrawImage*, o tempo de execução não diminuía como seria de esperar (devido à compilação *Just-In-Time* - ver Capítulo 3). Face aos resultados obtidos, pode-se concluir que o uso da função *BitBlt* é preferível face à função *DrawImage*. Logo, a solução híbrida é uma opção a considerar para a resolução deste problema.

Contudo, esta solução arrasta outro problema: qual o *overhead* associado com a interoperação entre *managed code* e *unmanaged code*? Se o *overhead* associado com a interoperação for significativo quando comparado com o ganho de desempenho obtido pelo uso das funções gráficas em ambientes exclusivamente *unmanaged*, então a solução híbrida deixa de ser vantajosa.

Segundo a *Microsoft* [Microsoft, 2005], o *overhead* médio associado com a transição entre código *managed* e *unmanaged* usando o *Platform Invoke* é de cerca de dez instruções máquina na *.NET Framework* (não existem dados oficiais sobre a *.NET Compact Framework*). Adicionalmente, é ainda referido que dependendo da frequência das chamadas efectuadas a *unmanaged code*, o *overhead* associado com estas chamadas pode variar entre valores negligíveis a valores inaceitáveis. É tambem referido que o *overhead* associado com este mecanismo também é influenciado pelo tipo de parâmetros utilizados. Embora o mecanismo envolva várias etapas (como se pode ver na figura 6.1), o *marshalling* dos parâmetros pode ter um efeito significativo no desempenho deste mecanismo. Assim, é necessário ter em atenção, os parâmetros utilizados. Estas afirmações deixam antever que a utilização do mecanismo *Platform Invoke* pode denegrir substancialmente o desempenho da aplicação. Extrapolando estas afirmações para o caso particular da aplicação *MapAdventure* podemos concluir que esta implementação seria arriscada, já que o acesso a *unmanaged code* seria efectuado sempre que o mapa fosse desenhado.

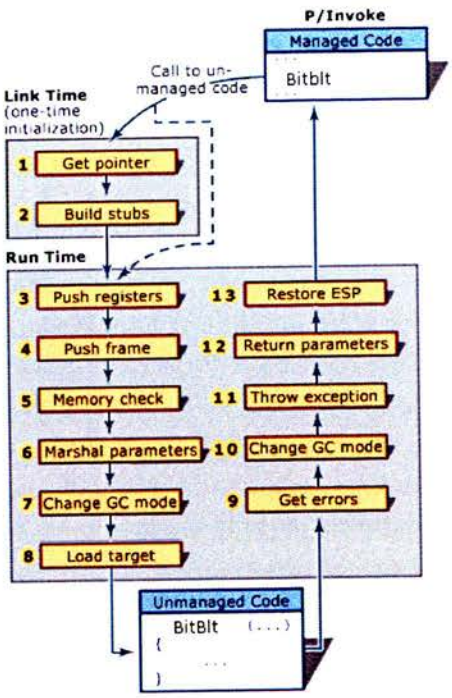

Figura 6.2 – O mecanismo *Platform Invoke*.

Dado que não existem dados oficiais sobre o *overhead* associado à utilização do mecanismo *P/Invoke* na plataforma *.NETCF*, foram efectuados testes adicionais. Estes pretendem, de um modo teórico, calcular o *overhead* associado ao mecanismo *P/Invoke*. Assim, foram efectuados testes adicionais que visam calcular o desempenho da função gráfica *BitBlt* num ambiente puramente *managed*. A diferença entre os valores obtidos com estes testes e o valores referentes à utilização do mecanismo *P/Invoke* dos testes anteriores, será, teoricamente, o *overhead* associado ao mecanismo *P/Invoke* Deste modo, foi construída uma aplicação, em *C++* (e utilizando a ferramenta computacional *eMbedded Visual C++ 4.0*), que utiliza a função *BitBlt*. Na figura 6.3 estão representados os resultados de todos os testes efectuados.

Como se pode verificar através do gráfico da figura 6.3, a diferença de desempenho entre o ambiente exclusivamente *unmanaged* (cor azul) e híbrido (cor amarela) é bastante significativa. Assim sendo, podemos concluir, de um modo teórico, que o *overhead* associado ao mecanismo *P/Invoke* não pode ser negligenciado e tem grande influência no desempenho da aplicação. Nos testes efectuados, este factor influenciou em cerca de 20% o desempenho gráfico da aplicação. Este valores são particularmente significativos para aplicações como o *MapAdventure* que são caracterizadas por um elevado nível de interacção com o utilizador onde a qualidade da aplicação se traduz, em grande parte, num tempo de resposta o mais pequeno possível. Neste contexto, torna-se vantajoso apostar na qualidade do desempenho da aplicação em detrimento de um rápido ciclo de desenvolvimento.

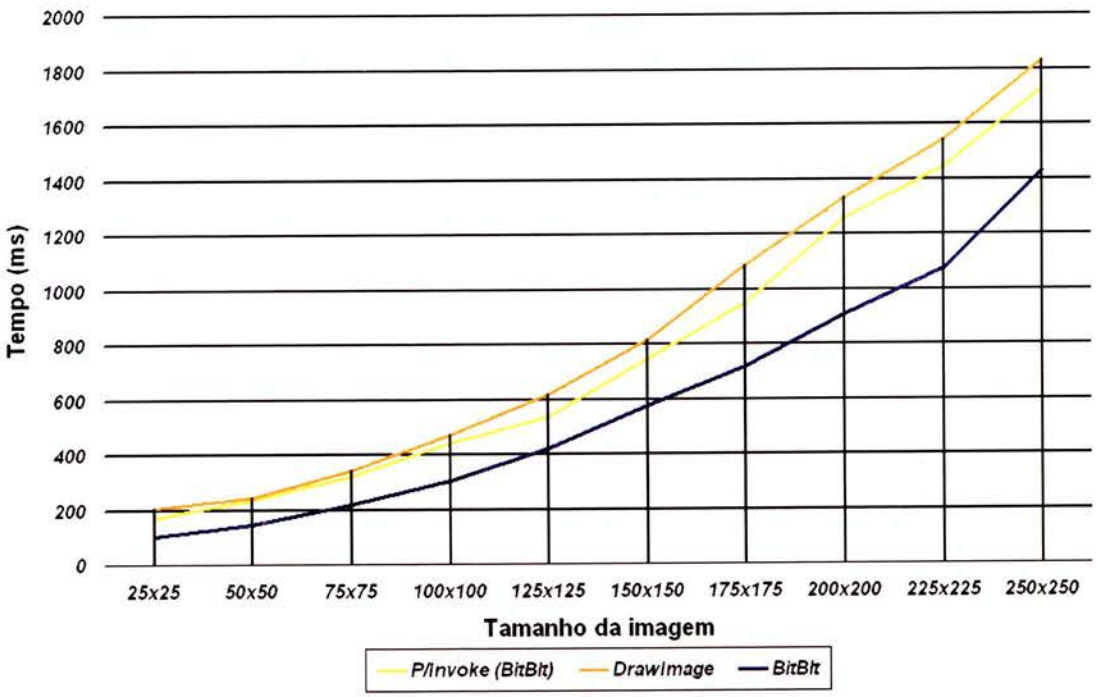

Figura 6.3 – Desempenho das funções gráficas *DrawImage*, *BitBlt* (com o mecanismo *PInvoke*) e *BitBlt* em ambiente exclusivamente *unmanaged*.

Assim, a sugestão do autor remete para a utilização dos métodos até agora utilizados na *ParadigmaXis*, pois estes conseguem obter um desempenho que actualmente não é conseguido na actual versão da plataforma *.NETCF*. No entanto, espera-se que em futuras versões da plataforma *.NETCF* seja possível obter níveis de desempenho próximos aos obtidos em linguagens como o *unmanaged C++*.

Nota: Os valores dos resultados dos testes efectuados às funções gráficas podem ser consultados no Apêndice A.

6.3 Resumo e Conclusões Finais

A fase inicial do estágio revelou-se crucial, pois permitiu a fácil adaptação do estagiário à metodologia de desenvolvimento seguida na empresa. Esta fase de integração decorreu de um modo bastante agradável, devido, essencialmente, ao bom ambiente de trabalho existente na empresa. Em termos práticos, esta fase foi principalmente dedicada conhecimento das tecnologias. No entanto, a aproximação à metodologia de *eXtreme Programming* levou a que, desde cedo, a fase de implementação fosse contemplada pelo estagiário.

Durante o decorrer do projecto, mas sobretudo durante a fase inicial, foi levado a cabo um trabalho de recolha de informação sobre as tecnologias envolvidas no desenvolvimento do sistema. A aprendizagem das tecnologias subjacentes à implementação dos motores gráficos desenvolvidos pela *ParadigmaXis*, mostrou-se bastante motivante, não só pelo interesse pessoal pela área relacionada, como também pelas potencialidades que estas tecnologias

demostraram possuir. A aquisição de conhecimentos na área relativa às plataformas *.NET* revelou-se crucial para o desenvovimento do projecto, o que demostrou que esta fase é essencial para a implementação de projectos a funcionar no mundo real, onde é necessário prever de um modo concreto quais as condicionantes e características do sistema a implementar. Por outro lado, esta fase proporcionou uma experiência bastante enriquecedora para o estagiário, pois permitiu o aprofundamento de conhecimentos nesta área, uma vez que este só dominava de um modo superficial as tecnologias *.NET*.

Ao longo de todo o projecto as tecnologias *.NET* proporcionaram um ambiente de desenvolvimento bastante eficiente. A vantagem na utilização destas tecnologias reside, essencialmente, na elevada usabilidade do ambiente de desenvolvimento integrado *Microsoft Visual Studio .NET*. Esta ferramenta permitiu que o desenvolvimento da aplicação *MapAdventure Studio* de pocessasse a um nível bastante rápido, mesmo na fase inicial em que o estagiário ainda se encontrava numa fase de adaptação à tecnologia.

Por outro lado, o desenvolvimento da aplicação *MapAdventure* permitiu chegar à conclusão que, no campo dos dispositivos móveis, a tecnologia *.NET* nem sempre é a escolha mais acertada. Antes de se optar por esta tecnologia devem ser analisados detalhadamente os requisitos do sistema a desenvolver face às funcionalidades oferecidas/omitidas pela *.NETCF* de modo a fazer uma escolha acertada. No caso particular da aplicação *MapAdventure*, a plataforma *.NETCF* não é aconselhada, pois não oferece os níveis de desempenho esperados ao nível gráfico. Neste caso, recomenda-se a plataforma de desenvolvimento *eMbedded Visual C++*, já que oferece um ambiente de maior desempenho e várias ferramentas de apoio ao desenvolvimento de alicações para os dispositivos móveis.

Embora já se esperasse que esta plataforma não atingisse os níveis de desempenho obtidos actualmente pela empresa, pretendia-se explorar ao máximo a plataforma de modo a tirar algumas conclusões sobre a viabilidade da sua utilização.

O facto do estágio envolver, simultaneamente, o desenvolvimento de uma aplicação comercial e o estudo de tecnologias recentes tornaram o estágio duplamente atractivo e bastante interessante para o estagiário. Por outro lado, permitiu ao autor adquirir experiência prática sobre o mercado de trabalho, onde as responsabilidades envolvidas são acrescidas. No fim, pode-se afirmar que os principais objectivos do estágio foram atingidos e que este se desenrolou sem percalços relevantes.

Bibliografia

[Beck, 2000] Beck, K. (2000). *Extreme Programming Explained.* Addison-Wesley.

[Burrough, 1986] Burrough, P.A (1986). *Principles of Geographical Information Systems.* Clarendon Press, Oxford.

[Câmara, 1996] G. Câmara, C.B.Medeiros, M.A.Casanova, A.Hemerly, G.Magalhães. Anatomia de Sistemas de Informação Geográfica (1996). Escola de Computação, SBC.

[Chandler et al., 2001] Chandler D., Fötsch M. (2001) *Windows 2000 Graphics API Black Book.* Coriolis Technology Press

[Chang, 2002] Chang K. (2002) *Introduction to geographic information systems.* McGraw Hill.

[Coulange, 1998] Coulange, B. (1998). *Software Reuse.* Springer-Verlag.

[Demers, 1997] Demers, M.N. (1997). *Fundamentals of geographic information systems.* John Wiley and Sons.

[DotGNU, 2004] DotGNU (2004). *DotGNU Project — GNU Freedom for the Net.* http://www.gnu.org/projects/dotgnu.

[Fox et al, 2003] Fox D., Box J. (2003). *Building Solutions with the Microsoft .NET Compact Framework: Architecture and Best Practices for Mobile Development.* Addison Wesley

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns — Elements of reusable object-oriented software.* Addison-Wesley.

[Graham, 2001] Graham, I. (2001). *Object-Oriented Methods — Principles & Practices (3rd Edition).* Addison-Wesley.

[Heywood et al., 1998] Heywood, I., Cornelius, S., Carver, S. (1998). *An introduction to geographical information systems.* Pearson Education Limited, Essex.

[James, 2000] James B-Y. Tsui (2000). *Fundamentals of Global Positioning System Receiver: A Software Approach.* John Wiley & Sons

[Jonh et al., 2003] John E. Harmon, Steven J. Anderson (2003). *The Design and Implementation of Geographic Information Systems.* John Wiley & Sons

[Knuth, 1984] Knuth, D. E. (1984). Literate programming. *The Computer Journal*, 27(2):97–111.

[Liberty, 2003] Liberty, J. (2003). *Programming C# (Third Edition)*. O'Reilly & Associates.

[Marques and Pedroso, 2002] Marques, P. and Pedroso, H. (2002). *C# Curso Completo*. FCA - Editora de Informática. http://www.fca.pt.

[Microsoft, 2005] Microsoft (2005) *An Overview of Managed/Unmanaged Code Interoperability* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/manunmancode.asp

[MONO, 2005] *MONO* (2005). *MONO Framework*. http://www.go-mono.org.

[Newkirk et al., 2005] Newkirk, J. W., Vorontsov, A. A., Two, M. C., Craig, P. A., and Poole, C. (2004). *NUnit Framework*. http://www.nunit.org.

[Opennetcf, 2005] *OpenNETCF* (2005). *OpenNETCF.org Smart Device Framework*. http://www.opennetcf.org

[Pick, 2004] Pick, James B. (2004). *Geographic Information Systems in Business*. Hershey, PA: Idea Group Publishing.

[PocketGPS, 2005] *PocketGPS* (2005). *PocketGPSWorld* http://www.pocketgps.co.uk/.

[Pocketpcmag, 2005] *PocketPC* (2005). *PocketPC Magazine* http://www.pocketpcmag.com/

[Richter, 2002] Richter, J. (2002). *Applied Microsoft .NET Framework Programming*. Microsoft Press.

[Rumbaugh et al., 1999] Rumbaugh, J., Jacobson, I., and Booch, G. (1999). *The Unified Modeling Language Reference Manual*. Addison-Wesley / ACM Press.

[Succi and Marchesi, 2001] Succi, G. and Marchesi, M. (2001). *Extreme Programming Examined*. Addison-Wesley.

[Thai et al., 2001] Thai T., Lam H. Q., (2001) *.NET Framework Essentials*. O'Reilly Associates

[TortoiseCVS, 2005] *Tortoise CVS* (2004). *Tortoise CVS* http://www.tortoisecvs.org/index.shtml.

[Totalpda, 2005] *TotalPDA* (2005). *TotalPDA*. http://www. totalpda.co.uk/.

[Vesperman, 2003] Vesperman, J. (2003). *Essential CVS*. O'Reilly & Associates, Inc.

[Wigley et al., 2003] Wigley A., WheelWright S. (2003). *Microsoft .NET Compact Framework Core Reference*. Microsoft Press

Apêndice A

Resultados dos testes

É objectivo deste anexo apresentar os resultados dos testes efectuados às funções gráficas. Na figura A.1 são apresentados os vários valores obtidos nos referidos testes.

Tamanho	25x25	50x50	75x75	100x100	125x125	150x150	175x175	200x200	225x225	250x250
BitBlt	96	138	202	292	418	630	780	937	1123	1399
	115	185	246	325	416	594	782	838	1199	1347
	92	154	219	279	425	546	686	881	1001	1415
	109	135	221	312	439	538	684	912	989	1467
	100	136	205	296	413	573	704	895	1102	1389
	105	141	228	283	417	580	762	895	1002	1489
	104	137	206	291	468	571	674	988	1089	1435
	115	146	207	313	395	571	707	974	1112	1439
	104	145	224	316	426	575	709	849	997	1498
	110	142	240	350	396	575	680	899	1123	1415
Média	**105**	**145,9**	**219,8**	**305,7**	**421,3**	**575,3**	**716,8**	**906,8**	**1073,7**	**1429,3**
P/Invoke e BitBlt	160	225	317	420	558	739	930	1230	1418	1689
	173	223	319	405	513	784	953	1243	1469	1701
	189	252	320	402	523	709	881	1163	1414	1734
	155	235	282	445	512	777	961	1311	1449	1756
	189	268	325	553	510	750	919	1377	1457	1695
	168	236	327	423	551	726	1164	1303	1477	1779
	176	235	335	488	555	686	928	1293	1460	1734
	163	255	413	427	594	740	904	1271	1429	1725
	181	223	288	446	540	777	918	1249	1479	1767
	172	229	316	401	531	784	940	1141	1414	1723
Média	**172,6**	**238,1**	**324,2**	**441**	**538,7**	**747,2**	**949,8**	**1258,1**	**1446,6**	**1730,3**
DrawImage	190	236	335	487	588	811	1154	1300	1528	1819
	202	248	346	500	595	853	1125	1318	1575	1853
	205	254	362	406	623	892	1115	1320	1527	1902
	201	236	326	478	609	751	1101	1289	1526	1714
	216	246	357	469	592	825	1086	1386	1618	1751
	190	244	356	463	604	784	1099	1249	1549	1886
	198	252	329	473	640	722	1012	1373	1481	1729
	234	249	334	512	726	898	1143	1307	1579	1799
	210	241	359	503	589	799	1071	1382	1496	1912
	202	249	321	442	614	830	983	1409	1545	1978
Média	**204,8**	**245,5**	**342,5**	**473,3**	**618**	**816,5**	**1088,9**	**1333,3**	**1542,4**	**1834,3**

Tabela A.1 – Valores dos testes efectuados às funções gráficas