UNIVERSIDADE FEDERAL DE PERNAMBUCO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

UM *FRAMEWORK* PARA DESENVOLVIMENTO DE APLICATIVOS EM *WINDOWS MOBILE*.

TRABALHO DE GRADUAÇÃO

Aluno: Bruno Costa Bourbon (bcb@cin.ufpe.br)
Orientador: Prof. Dr. André Luis de Medeiros Santos (alms@cin.ufpe.br)

Recife, Agosto de 2005.

UNIVERSIDADE FEDERAL DE PERNAMBUCO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

UM *FRAMEWORK* PARA DESENVOLVIMENTO DE APLICATIVOS EM *WINDOWS MOBILE*.

ESTE TRABALHO FOI SUBMETIDO À GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA CONCLUSÃO DA DISCIPLINA DE TRABALHO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE E OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR: PROF. DR. ANDRÉ LUÍS M. SANTOS

Recife, Agosto de 2005.

Resumo

Num crescente mercado de celulares, operadoras e fabricantes buscam adicionar novos recursos em celulares de forma a obterem e manterem clientes, e com a miniaturização dos componentes celulares passaram de simples aparelhos de comunicação de voz para ferramentas de entretenimento e trabalho. Com objetivo de diminuir o tempo de desenvolvimento de software para ambiente móvel, ferramentas e processos foram, e estão sendo adaptados. Uma solução na etapa de implementação desse software é a utilização de *frameworks*, que podem facilitar o reuso de partes do código e também do design dos softwares.

O objetivo deste trabalho é fazer um estudo sobre a portabilidade de um *framework* de desenvolvimento de aplicações já existente na plataforma QUALCOMM BREW para o sistema operacional *Windows Mobile 2003 Second Edition* da Microsoft.

Palavras chaves: Windows Mobile, framework, aparelhos celular, portabilidade.

Dedicatória

Este trabalho é dedicado aos meus avôs João e Angelita(*in memorian*), Geraldo(*in memorian*) e Anita, por ensinarem aos meus pais Juscelino e Marisa, o valor de uma boa educação.

Agradecimentos

Agradeço primeiramente aos meus pais, meus irmãos, tios e tias, primos e primas, que sempre me deram apoio em minha vida e por me aturarem durantes minhas fases de irritação por decepções.

Em especial meu orientador professor André Santos, por acreditar depois de falhar uma vez , eu me levantaria e conseguira. Os professores do centro de informática por seus conhecimentos.

A Tiago Barros por deixar utilizar parte de seu projeto de mestrado como tema para este trabalho.

Agradeço também em especial a equipe DELTA do C.E.S.A.R., Gilberto Alves, Magda Barros, Emerson Espínola, que trabalharam comigo e Tiago na versão BREW. Hugo, Dio, Leila, Roberio, Rogério, Gledson e Victor, que estão e estiverem comigo em outros projetos, ao indiano Suresh, que apesar do pequeno estágio em nossa equipe, ajudou na conclusão da aplicação de teste. E nossos *Technical Leaders* Fred Monteiro e Gustavo de Paula.

Aos amigos de universidade, por sofrerem comigo, durante quatro anos e meio, pelas festas e pelas madrugadas realizando projetos nos laboratórios.

E meu eterno agradecimento a Thaís, amiga e namorada, pela paciência e amor demonstrados durantes todos esse anos.

Sumário

1. Introdução 1
2. Plataformas de Desenvolvimento 4
2.1. J2ME 4
2.2. BREW 8
2.3. *Symbian OS* 11
2.4. Windows Mobile 14
2.5. Comparação 16
3. Framework e o CMF 17
3.1. Uso de *Frameworks* 17
3.2. C.E.S.A.R. *Mobile Framework* 21
4. Windows Mobile™ 26
4.1. História e Versões 26
4.2. Estrutura 28
4.3. Ferramentas 29
4.4. Desenvolvimento 31
4.5. Conclusões 37
5. Adaptando CMF para *Windows Mobile* 2003 39
5.1. *BREW* vs. *Windows Mobile 2003* 39
5.2. Estudo de Soluções 41
5.3. Adaptando o Código do *Framework* 41
5.4. Adaptando as Funcionalidades do *Framework* 42
5.5. Viabilidade 43
6. Conclusão 44
6.1. Dificuldades Encontradas 44
6.2. Trabalhos Futuros 45
7. Referências 46

Índice Figuras

Figura 1. Modelo da Arquitetura Java 5
Figura 2. Representação da Arquitetura *BREW*..... 9
Figura 3. Visão Funcional do Symbian OS v7.0..... 14
Figura 4. Plataforma Pocket PC (Smartphone). 15
Figura 5. Visão da Arquitetura do CMF. 22
Figura 6. Evolução do *Windows Mobile* 27
Figura 7. Estrutura do *Windows Mobile 2003*..... 28
Figura 8. embedded Visual C++. 30
Figura 9. Visual Studio .NET 2003 30
Figura 10. Visual Studio 2005 31
Figura 11. Exemplo de um programa *Windows Mobile for Smartphone*. 37

Índice Tabelas

Tabela 1. Comparação das Principais Características de J2ME / BREW / Symbian / Windows Mobile 16

1. Introdução

No inicio os celulares eram apenas versões sem fio e de longo alcance dos aparelhos telefônicos residências, devido ao estado da tecnologia (e seus custos) os celulares não fazia nada além de receber e gerar ligações de voz. Com a introdução dos aparelhos de 2ª geração e dando continuidade à lei de Moore[01], o aparelhos tiverem-se sua capacidade aumentada permitindo-se que novas funcionalidades fossem acrescidos a sua principal atividade gerar e receber ligações. Aplicações como agendas telefônicas e compromisso, calculadora, jogos, acesso a Internet, entre outros. Mas no inicio somente as fabricantes faziam os programas que viam embutidos nos celulares e eram fechados, não permitindo que novas aplicações fosse colocadas nos aparelhos, obrigando ao usuário a mudar de aparelho todas vez que quisessem uma nova aplicação.

Vendo o problema do ponto de vista do usuário, os fabricantes notaram uma nova oportunidade de negócios, celulares que poderiam adicionar novas funcionalidades depois de sair da fabrica. Em 2001 a Motorola lançou o primeiro aparelho com essa capacidade introduzindo no mercado um versão móvel da plataforma Java[02], dando inicio a uma nova revolução a telefonia móvel e ao mundo da tecnologia da informação. Apesar dos PDAs (*Personal Digital Assistants* – Assistentes Digitais Pessoais) terem iniciado o mercado de computação móvel eles não tiveram e nem têm a penetração que os celulares alçaram.

Com o crescimento do mercado e variedade de aparelhos e plataformas os desenvolvedores de software esbarraram-se com algumas dificuldades, sendo as principais o *time-to-market* (o momento que aplicação deve estar no mercado antes que fique ultrapassado ou que aparecem vários concorrentes) [03], e o *porting*, adaptação de um programa para outras plataformas (hardware ou software) de forma a atingir o maior número de usuários possível.

Em vista disso o mestrando do Centro de Informática da Universidade Federal de Pernambuco e Engenheiro de Sistemas do C.E.S.A.R. (Centro de Estudos de Sistemas Avançados do Recife) Tiago Barros inspirado por trabalhos na área de desenvolvimento de jogos para aparelhos móveis, propôs a criação de um padrão aberto para o desenvolvimento (*framework*) de aplicativos (programas de forma geral) que possa diminuir o tempo (e conseqüentemente os custos) para *porting* de um aplicativo para várias plataformas. Atualmente sendo desenvolvido para a plataforma Qualcomm BREW[04].

Nossa proposta é verificar a portabilidade desse *framework* para plataforma Microsoft *Windows Mobile*[05]. De maneira a validar esse *porting* a mesma aplicação usada para testar o *framework* em BREW será usada como referência, de forma que possamos sugerir mudanças, ajustes, ou até mesmo repensar a arquitetura desse *framework*.

Em interessante esclarecer o uso de certos termos, principalmente, *porting* ou portar, que a técnica para implementação de um sistema ou aplicativos que já foi desenvolvido em um sistema operacional (ou ambiente de programação) para outro, ou mesmo adaptação do aplicativo em outro hardware (com mesmo OS). *Smartphone* é um celular com capacidade processado avançado que permite a utilização de aplicativos típicos de computadores de mesa (*desktops*) ou PDAs. E *wireless* (do inglês, sem-fio) que nesse trabalho refere-se (salvo dito em situação especificas) à rede celular ou os celulares propriamente dito.

Esse trabalho no seu capitulo dois é visto o quê é uma plataforma de desenvolvimento, as principais plataformas utilizadas no mercado mundial de aplicativos celulares, e uma comparação dessas plataformas. No Capitulo três mostramos o que é o *framework* desenvolvido por Tiago Barros, explicado quais as vantagens e desvantagens de utilizar *frameworks* para desenvolvimento de aplicações móveis. No Capitulo quatro entramos em detalhe da plataforma *Windows Mobile*, um pouco de sua história, sua estrutura, ferramentas, e com é desenvolvido uma

aplicação nessa plataforma. No capitulo cinco veremos que ajustes necessários para que o *framework* funcione na Plataforma *Windows Mobile*, validando e discutindo as características do *framework*. No capitulo seis são apresentados as conclusões sobre a adaptação (*porting*), as dificuldades encontradas e sugestões para continuidade do *framework* e de futuros *portings*.

2. Plataformas de Desenvolvimento

As plataformas de desenvolvimento para celulares são ambientes, ferramentas, técnicas, e linguagens de programação que possibilitam a codificação, simulação (ou emulação), depuração e teste de programas voltados para aparelhos celulares habilitados para tal.

Como dito anteriormente todas as aplicações que vinham nos aparelhos, e que não podiam ser removidos, atualizados, ou inseridos, eram desenvolvidos pelos próprios fabricantes dos aparelhos, seja por equipe interna ou empresas terceirizadas, mas sempre para plataformas proprietárias. Mas isso mudou no momento que a *Sun Microsystem* [06] resolveu colocar a plataforma Java, dentro do celular e em parceria com a Motorola [07] lançou o primeiro aparelho comercial com tal tecnologia. Alguns podem dizer que o IBM *Simon* e o sistema operacional *Zaurus OS* poderiam ser a primeira plataforma de desenvolvimento para celulares, mas precisamos diferencias celulares com capacidade de PDAs e PDAs com capacidade de celulares. A principal e mais importante dessas diferencas é que um celular com funcionalidade de PDA é que ele tem como prioridade total o uso como comunicador de voz, enquanto PDAs com capacidade de celulares utilizam-se da rede móvel celular para transmissão e recebimento de dados.

Deixando esse debate de lado, precisamos entender um pouco mais sobre as plataformas de desenvolvimento mais usadas em aparelhos celulares. São elas:

2.1. J2ME

A *Java 2 Platform, Micro Edition* (*J2ME*)[08] provê um ambiente robusto e flexível para aplicações rodarem em produtos eletro-eletrônicos de consumo, tal como telefones celulares, *PDA*s, receptores de TV, bem como uma extensa linha de aparelhos embarcados. Como suas contrapartes para ambientes corporativos (*J2EE*), *desktop* (*J2SE*) e *smart card* (*Java Card*)[09], *J2ME*

incluem máquinas virtuais Java, e um conjunto de *API*s Java definidas através do *Java Community Process* [10](sistema de discussão criado pela Sun para ajudar na evolução do Java), por grupos especialistas cujo membros incluem os principais fabricantes de equipamentos, vendedores(desenvolvedores) de softwares, e provedores de serviços.

J2ME entrega o poder e os benefícios da tecnologia Java para equipamentos embarcados e de consumo. Ele inclui interfaces de usuários flexíveis, um modelo de segurança robusto, uma variedade de protocolos de rede, e extenso suporte para aplicações isoladas (*offline*) e conectadas (*networked*) que podem ser dinamicamente baixadas. Aplicações baseadas nas especificações *J2ME* são escritas uma vez e usadas numa grande variedade de dispositivos, também explorando as capacidades nativas de cada dispositivos.

2.1.1 A Arquitetura *J2ME*

A arquitetura J2ME define configurações, perfis e pacotes opcionais como elementos para construir ambientes completos de execução(*runtime*) de programas Java que atendem os requisitos de um larga variedade de dispositivos e mercados específicos. Cada combinação é otimizada para as capacidades de memória, poder de processamento, e *I/O* (Entrada e saída de dados). O resultado é uma plataforma Java comum que alavanca completamente cada tipo de dispositivo a levar uma rica experiência ao usuário.

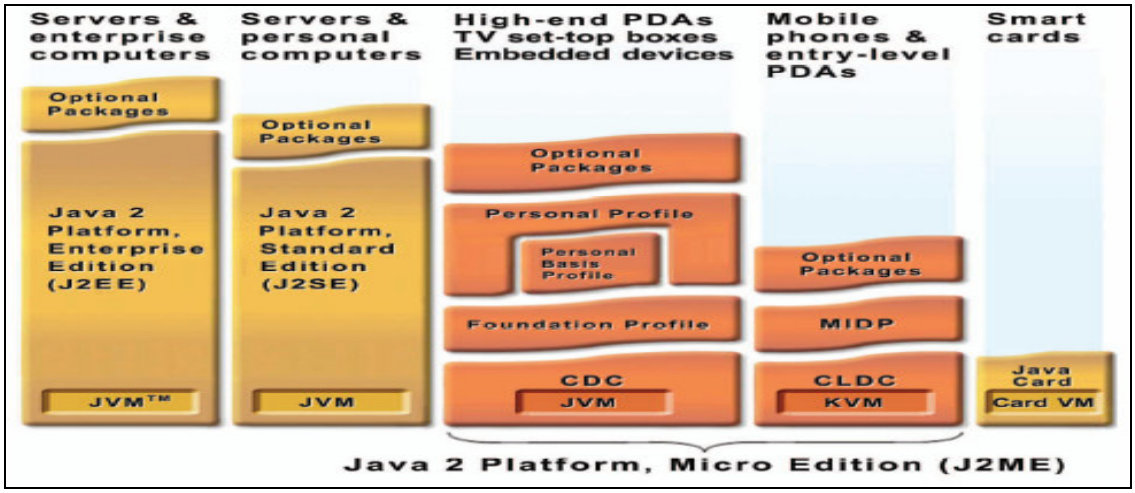

Figura 1. Modelo da Arquitetura Java

2.1.2 Configurações

Configurações(*Configurations*) são compostas de uma máquina virtual e um conjunto mínimo de biblioteca de classes. Elas provêem a funcionalidade básica para uma série particular de dispositivos que compartilham características em comum, como, por exemplo, restrições de conectividade e memória. Atualmente, existem duas configurações *J2ME*:

- *Connected Limited Device Configuration (CLDC)*: CLDC é a menor das duas configurações, projetada para dispositivos com conexões de rede intermitentes, processamento lento e memória restrita - aparelhos como celulares, *pagers* e *PDA*s. Esses equipamentos tipicamente possui *CPU*s de 16 ou 32 bits, e um mínimo de 128KB a 512KB de memória disponível para implementação da plataforma Java e aplicações associadas.
- *Connected Device Configuration (CDC)*: CDC é projetado para dispositivos que possuem mais memória, processadores mais rápidos, e conexão de banda-larga, com receptores de TV, *gateway*s residências, sistemas veiculares (de localização e entretenimento) e poderosos *PDA*s. CDC inclui uma máquina virtual Java completa, e com um subconjunto da plataforma *J2SE* maior que a CLDC. Como resultado, a maioria dos dispositivos CDC têm *CPU*s de 32 bits e um mínimo de 2MB de memória disponível para a plataforma e aplicativos associados.

2.1.2 Perfis

Em ordem a fornecer um ambiente completo de execução direcionado para categorias especificas de dispositivos, as configurações (*configurations*) devem ser combinadas com um conjunto de *API*s de um nível maior, ou perfil, que adicionalmente definem o modelo de ciclo de vida da aplicação, a interface do usuário, e o acesso à propriedades especificas do dispositivo.

- *Mobile Information Device Profile* (MIDP): é projetada para telefones celulares e *PDA*s básicos. Ele oferece a essência de funcionalidade de aplicação requerida para aplicações móveis, incluindo a interface de usuário, conectividade de rede, armazenamento local de dados, e gerenciamento de aplicativos. Combinado com o CLDC, MIDP provem um ambiente completo de execução Java que eleva a capacidade de dispositivos *handheld* (que podem ser segurados com uma mão) e minimizar o consumo de memória e energia.

- *Foundation Profile* (FP): perfis CDC estão em camada de maneira que perfis possam ser adicionados quando necessários a fornecer funcionalidades de aplicação para diferentes tipos de dispositivos. O perfil FP é o mais baixo nível de perfil para CDC. Ele fornece uma implementação CDC com capacidade de rede que pode ser usada por implementações fortemente embarcadas sem interface gráfica. Ele também combinar com o *Personal Basis Profile* e o *Personal Profile* para dispositivos que requerem uma interface gráfica com o usuário (GUI).

- *Personal Profile* (PP): é perfil CDC orientado para dispositivos que requerem Interface (GUI) completa ou suporte para *Applets* de Internet, com *PDA*s de última geração, aparelhos de comunicação e videogames. Ele inclui integralmente as bibliotecas *Java Abstract Window Toolkit* (AWT) e oferece fidelidade *Web*, executa facilmente *applets* baseado na *Web* projetados para uso em ambientes *desktop*. PP substitui a tecnologia *PersonalJava*™ e fornece um claro caminho para migração de aplicativos *PersonalJava* para a plataforma *J2ME*

- *Personal Basis Profile* (PBP): É um subconjunto de PP, que fornece ambiente de aplicação para dispositivos conectado a rede que suportam um nível básico de apresentação gráfica ou requerem o uso de componentes gráficos especializados para aplicações especificas. Os aparelhos incluem receptores de TV, sistemas veiculares, e quiosques de informação. Ambos PP e PBP estão em camadas sob do CDC e FP.

2.1.3 Pacotes Opcionais

A plataforma *J2ME* pode ser adicionalmente estendida pela combinação de vários pacotes opcionais com as CLDC, CDC e seus perfis correspondentes. Criado para atender requisitos de um mercado muito específico, pacotes opcionais oferecem *API*s que usam tanto tecnologias existentes e emergentes como por exemplo *Bluetooth*, *Web services*, Mensagens sem-fio, multimídia, e conectividade a banco de dados. Por serem modulares, fabricantes de dispositivos podem incluí-los quando necessário para integrar plenamente aspectos de cada dispositivo.

2.2. BREW

BREW, ou *Binary Runtime Environment for Wireless* (Ambiente Binário de Execução para Celulares), da QUALCOMM, é uma completa solução de ponta-a-ponta para o desenvolvimento de aplicações sem-fio (*wireless*), configuração de aparelhos, distribuição de aplicação, cobrança e pagamento. A solução completa *BREW* inclui o *BREW SDK* (do inglês Kit para Desenvolvimento de Software) para desenvolvedores de aplicativos, o software cliente *BREW* e ferramentas para o *porting* para fabricantes de celulares, e o BREW *Delivery System* [11](BDS - Sistema de Distribuição *BREW*) que é controlado e gerenciado pelas operadoras (de telefonia) - permitindo-lhes facilmente colocar aplicações de desenvolvedores no mercado e coordenar o processo de cobrança e pagamento. Serviços (baseados em BREW) das operadoras permitem a assinantes personalizarem seus aparelhos através do *download* (*online* e sem-fio) de aplicações de seus servidores de *downloads*.

Do lado do cliente ao lado do servidor, *BREW* é otimizado para o ambiente sem-fio e é construído para atender os desafios que as operadoras encontram na distribuição de serviços de dados sem-fio. Escolhendo as vantagens a inteira solução *BREW*, operadoras podem rapidamente empregar seus serviços de dados sem-fio - e, graças a arquitetura extensível de *BREW*, operadoras podem oferecer qualquer tipo de aplicação com facilidade.

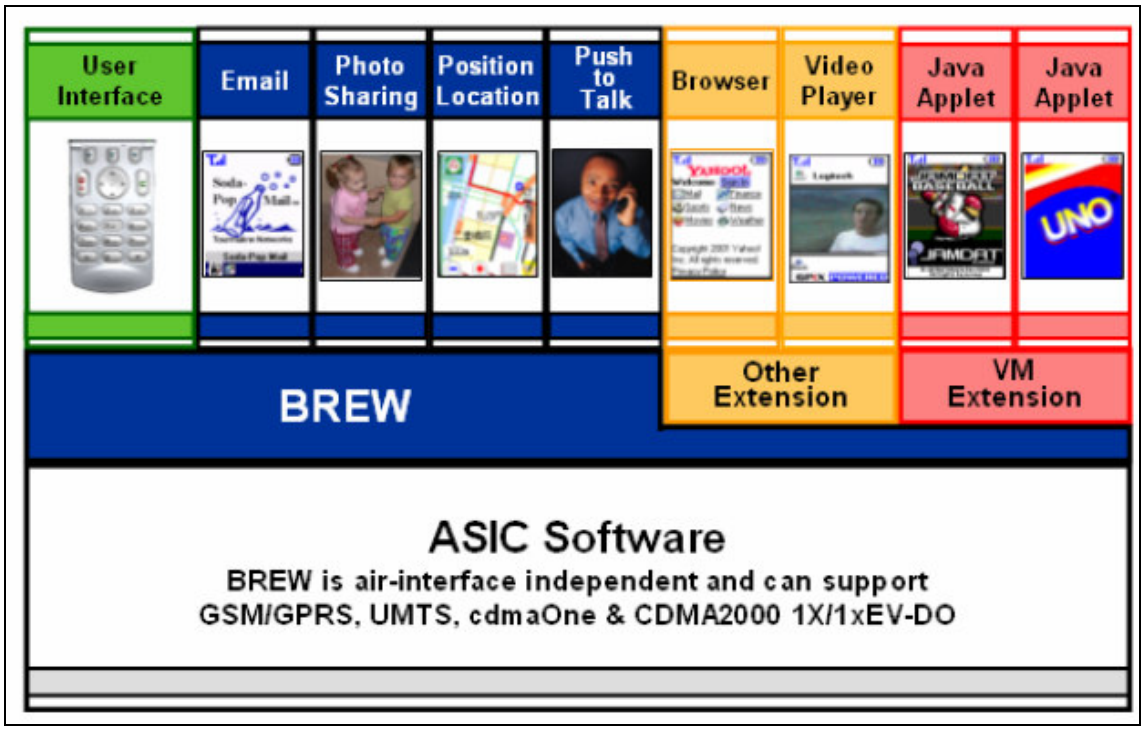

Figura 2. Representação da Arquitetura *BREW*

2.2.1 *BREW SDK*

Durante a elaboração deste trabalho, cinco versões do *BREW SDK* estão disponível para *download*: 1.0, 1.1, 2.0, 2.1 e 3.1 (a 3.0 foi cancelada). Dependo de quantos usuários deseja-se atingir é necessário baixar uma ou mais versões do SDK. O Desenvolvedor deverá prestar atenção na hora de construir uma aplicação para celulares que são habilitados para *BREW*, a responsabilidade de implementação da plataforma fica a cargo dos fabricantes, podendo eles habilitar ou não determinadas interfaces (serviços). A QUALCOMM junto com os fabricantes disponibiliza em seu site [12] informações sobre cada aparelho (as chamadas *DDS – Device Data Sheet*, Planilha de Dados de Equipamento). Segundo a propria QUALCOMM os a versões são reversamente compatível, ou seja, programas desenvolvidos na versão 1.1 funcionam na 2.0, 2.1 em diante. Mas isso não em bem verdade, pois da versão 2.1 para 3.1, segundo seu documentos em seu *site* [13], alguns interfaces foram depreciadas, ou substituídas, o que acabam quebrando a idéia de reversamente compatíveis. O *SDK* é composto por uma *API*, que nada mais

é que uma biblioteca de interface (ou funções) em linguagem C [14]. Entre as dezenas de funções encontras interfaces para:

- Acesso à agenda do celular;
- A *ringtones* (toques de celulares);
- Câmera (em celulares que possuem);
- Multimídia (vídeo, som, texto);
- Envio e Recebimentos de SMS, e até ligações telefônicas;
- Banco de Dados;
- Acesso a Internet
- Entre outros.

O desenvolvimento de aplicativos em *BREW* é um pouco mais complicado, é necessário possuir o Microsoft *Visual C++* 6.0 [15] ou *Visual Studio.NET*[16]. O *SDK* já conta com uma série de ferramentas que auxiliam a construção do aplicativo, um assistente em formado de *plug-in* para o *VC6* ou *VS.NET*, um editor de recursos, editor de informações de identificação e acesso (a determinados recursos do celular), um simulador e editor de capas (ou *skins*), que são imagens dos aparelhos, mais informações de comportamento do aparelho. O simulador não é muito eficiente, pois ele não recria com 100% de fidelidade o comportamento da aplicação dentro do aparelho (isso requereria a imagem da ROM de todo os aparelho, inviável econômica e tecnologicamente falando). Todo aplicativo *BREW* utiliza-se de quatros arquivos:

- Module (.*mod*) que é a aplicação em si;
- *BREW Application Resource* (.bar) que pode conter, imagens, diálogos, e textos (e binários na versão 3.x). Uma aplicação pode usar quantos arquivos “BAR” desejar (ou nenhum se não precisar);
- *Signature* (.*sig*) que é um tipo de certificado que permite o uso da aplicação num único celular (utiliza-se do número serial do aparelho);
- *Module Information File* (.mif) que contém informações básicas do programa com ícone (para o atalho), seu *ClassID*, e as permissões de acessos a certos recursos do celular;

2.2.2 *BREW Delivery Systems (BDS)*

Um dos aspectos mais interessantes de BREW foram sua API, que dá acesso ao desenvolvedor a quase todas as funcionalidades de um aparelho celular (depende do fabricante), é seu modelo de negócio. O modelo funciona de forma peculiar. No lado do desenvolvedor ou empresa desenvolvedora, a empresa deve se cadastrar junto a QUALCOMM como desenvolvedor autenticado (com custo de US$ 400,00 anuais) que dá o direito de gerar 200 assinaturas, cada aparelho deve ser enviado para a QUALCOMM para ser habilitado para o modo de teste (para que aplicativos sejam colocados no celular através de cabos de dados). Uma vez autenticados o desenvolvedor poderá ter acesso a vários recursos no *site* do *BREW*, como ferramentas para transferência dos aplicativos para celulares, ferramentas para depuração, e assinatura de aplicativo, além de documentos específicos (incluindo as *DDS*). Depois de testar e compilar para o celular (que exigem um *cross-compiler* especifico para *BREW*). As aplicações devem ser submetidas (pago) para certificação *True BREW*™[17], que indica que a aplicação não danificara o aparelho. Depois de vencida essa batalha cabe o desenvolvedor escolher o modelo de comercialização de seu aplicativo (que pode ter tempo limitado, licenças limitadas, ou ilimitadas), e a partir desse ponto cabe as operadoras venderem os aplicativos aos usuários finais. O *download* (compra ou assinatura) do serviço (aplicativo) é feito a partir de uma função especifica em cada aparelho, o usuário acessa um catálogo (tipo de pagina na Internet) de sua operadora e compra a aplicação que é enviada para seu aparelho. A cobrança é feita através do sistema *BDS* (que pode ser conectado ao sistema financeiro da operado) é automática repassa os valores acertados ao desenvolvedor.

2.3. *Symbian OS*

Até algum tempo atrás a maioria dos fabricantes de celulares usavam sistemas operacionais proprietários em seus produtos. Os celulares eram simples e não precisavam de sistemas operacionais complexos ou plataformas de desenvolvimento de software. Hoje em dia vemos que vários celulares são capazes de executar aplicações baixadas, e não apenas as pré-instaladas nos aparelhos por seus fabricantes. Esses novos dispositivos são cada vez mais

complexos e exigem sistemas operacionais mais complexos que garantam uma plataforma confiável e versátil para empresas de desenvolvimento de software.

Para criar tal sistema operacional, algumas das lideres na indústria de comunicação sem-fio criaram uma empresa privada independente, Symbian[18],em 1998. A empresa é de propriedade da Nokia, Motorola, Siemens, Sony Ericsson, Matsushita (Panasonic), Psion e Siemens. A Symbian é fornecedora de um sistema operacional aberto, avançando e padronizado para celulares chamado Symbian OS[19]. O Symbian OS é baseado no sistema operacional EPOC da Psion.

Para entendermos a diferença entre J2ME e o Symbias OS é que em no primeiro caso é ambiente de execução de aplicativos que utilizam a infra-estrutura do celular (incluindo o sistema operacional) como base para rodas as chamadas máquinas virtuais, mas esse tipo de abstração, que garantem um alto grau de portabilidade das aplicações, perde no acesso a recursos específicos como câmera, agenda telefônica, entre outros itens, apesar de isso estar sendo compensado nas novas versões. Para compensar essas deficiências desenvolvedores devem programar os aplicativos no sistema nativo dos celulares, e aí que entra o Symbian OS. Como sistema operacional e plataforma de desenvolvimento as empresas de software tem um ambiente completo para aplicações móveis que traz recursos poderosos aos aplicativos e, como está vários modelos de vários fabricantes de celulares, garante uma boa portabilidade dos aplicativos.

2.3.1 Características Principais

O Symbian OS já se encontra na versão 9.1 lançada recentemente. Mas preferimos dar destaque à versão 7.0 que é utilizada pela famosa série 60 [20] da Nokia []. Na versão 7.0 as principais características são[21]:

- Rica Suíte de Motores de Aplicativos: Contatos, Agenda, Mensagem, Navegação, Office, utilitários e controle de sistema; OBEX para troca de objetos – por exemplo cartões de visita; *APIs* integradas para gerenciamento de dados, texto, área de transferência e gráficos.

- Navegador – *Engine* (motor ou *framework*) de navegação para completo acesso a *web* e com suporte a *WAP* (protocolo para aparelhos móveis, equivalente ao HTTP).
- Mensagem – Detalhado ambiente integrado para mensagens usando MMS, SMS, e EMS (protocolos de mensagem entre celulares); E-mail de Internet usando POP3, IMAP4, SMTP, MHTML (protocolos de acesso a e-mail); Anexos padronizados; Fax.
- Multimídia – Acesso compartilhado de tela, teclado, fontes e *bitmaps* (imagens onde cada ponto é representado por um ou mais bytes de informação); Gravação e reprodução de áudio, e funcionalidades ligadas a imagens (suportando os mais comuns formatos de áudio e imagem), incluído *API* de aceleração gráfica, *streaming* (envio de informações em blocos que são exibidos a medida que chegam), e acesso direto a tela.
- Protocolos de Comunicação – Camadas para redes *WAN*. incluindo TCP, IPv4, IPv6 (com IPSEC) e WAP, camadas para redes locais incluído infravermelho, tecnologia sem-fio *Bluetooth* e *USB*.
- Telefonia móvel – Symbian OS inclui um *API* abstrata para padrões de celulares 2G, 2.5G e 3G, permitindo a fabricantes levar o telefones Symbian OS, para qualquer parte do mundo.
- Suporte a Internacionalização – suporte nativo a *Unicode* (formato em que cada caractere é representado por 2 bytes em vez de 1 do formato *ASCII*), *framework* flexível para entrada de texto, e formatação e fontes adicionais de texto.
- Sincronização de Dados – Sincronização *Over-the-air* (OTA, do inglês Sobre-o-Ar, indicando que a transferência sem-fio) suportado utilizando SyncML . Sincronização com PC através de kits, *Framework* fornece sincronização para dados *PIM* (do inglês Gerenciamento de Informação Pessoal), transferência de dados, e conversão de documentos.
- Segurança – Completa encriptação e gerencia de certificados, protocolos de comunicação segura (incluindo HTTPS, WTLS e SSL), *framework* WIM (gerencia de informação sem-fio) e instalação baseada em certificação.
- Desenvolvimento de Software – Opções de desenvolvimento de conteúdo e programação para aplicações móveis e serviços de

desenvolvimento: C++, Java (J2ME MIDP 1.0 e Personal Java 3.0 com JavaPhone 1.0), WAP e web.

- Suporte a múltiplas interfaces de usuários – Qualquer mecanismo de entrada de teclados normais, tela de toques, a teclados numéricos de telefone.

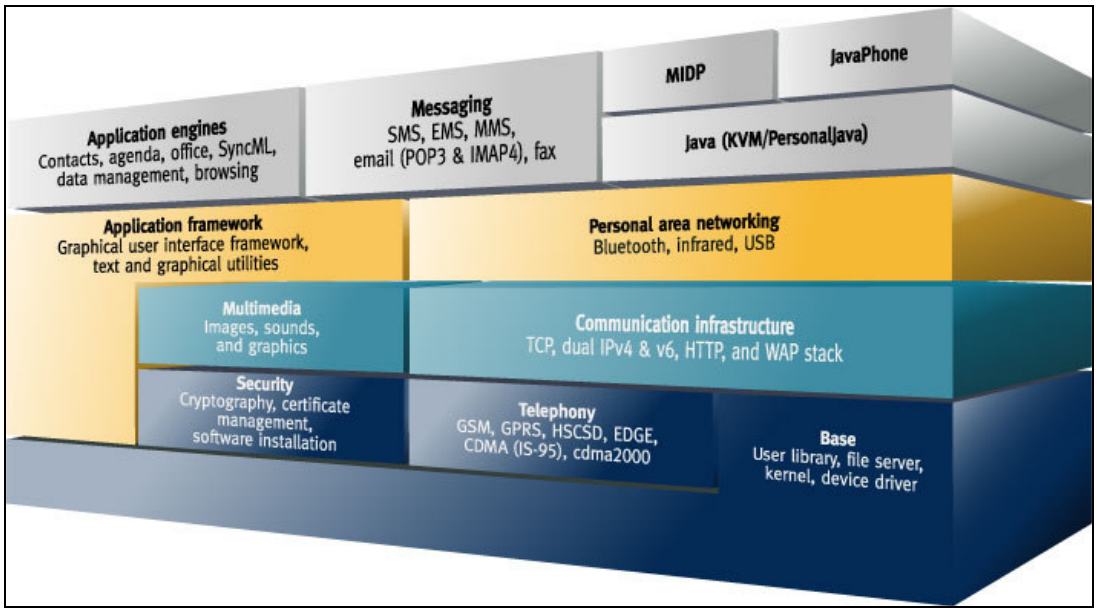

Figura 3. Visão Funcional do Symbian OS v7.0

2.4. Windows Mobile

A Microsoft anunciou em 2003 a criação do *Windows Mobile*[22], uma nova marca global de softwares Microsoft para dispositivos móveis por exemplo *Pocket PCs* e *Smartphones*. A nova marca *Windows Mobile* ajuda a consumidores à prontamente entender a experiência consistente de uso que eles podem esperar dos softwares dentro de *Pocket PCs* e *Smartphones*. A nova marca também reflete o compromisso, da Microsoft ao mundo móvel em levar seus softwares para dispositivos móveis para a família Windows.

O *Windows Mobile* 2003 para o *Pocket PC* (e *Smartphone)* é construído sobre o Windows CE (um sistema operacional componentizado destinado ao mercado de sistemas embarcados) adicionando-se nova funcionalidades, interface com usuário, e aplicações para criar uma plataforma computacional móvel otimizada para dispositivos de mão (celulares e *PDAs*).

Especificamente o *Windows Mobile* é baseado *no Windows CE .NET 4.2*. A figura a seguir exibe o relacionamento entre o sistema Windows CE e a plataforma *Windows Mobile*.

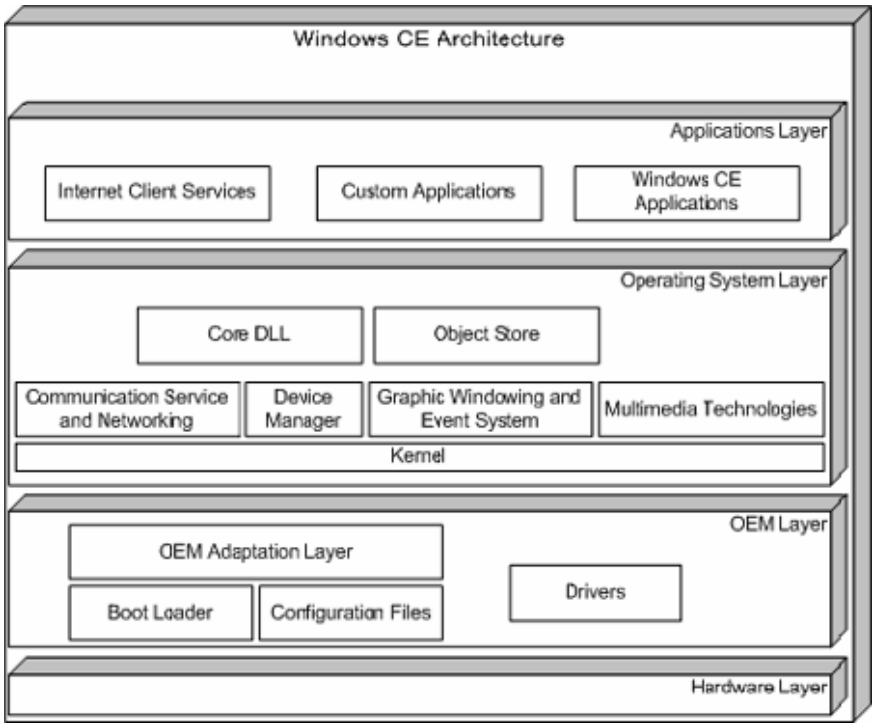

Figura 4. Plataforma Pocket PC (Smartphone).

Windows Mobile 2003 para Pocket PC permite aos principais fabricantes dispositivos móveis inovadores através da otimização a interface com usuário, aplicações e correspondentes conjuntos de atributos em torno de cenários de conectividade e gerenciamento de informação pessoal. Por padronizar os requisitos básicos de hardware e prover um conjunto consistente de *APIs* de programação, o *Windows Mobile 2003* fornece um ambiente de desenvolvimento de aplicação entre plataformas (de hardware). Essa consistência permite Essa consistência permite a comunidade externa de desenvolvimento construir aplicações que objetivam eficientemente uma base singular de consumidores. O Kit de Desenvolvimento de Software para *Windows Mobile 2003* para *Pocket PC* e ambiente de emulação além de autorizar a grande comunidade de desenvolvedores Windows criar aplicações terceirizadas para *Windows Mobile*, aumentam sua atração como uma plataforma para a comunidade desenvolvedora. E como resultado, um conjunto rico de aplicativos vem sendo criado para essa

plataforma, ajudando a guiar a inovação de hardware e periféricos, suportados pelos fabricantes lideres de mercado e a congregação de vendedores independentes de hardware e software, integradores de sistemas, provedores de soluções móveis, clientes corporativos, e operadoras móveis.

2.5. Comparação

Neste capitulo vimos as quatros principais plataformas de desenvolvimento de software para sistemas móveis. De forma a comparar as principais características dessas plataformas a pequena tabela abaixo pode servir de referência.

	J2ME/MIDP 2.0	**Symbian OS**	**BREW**	**Window Mobile 2003**
Tamanho permitido para as aplicações	Poucos kilobytes (em geral, 64KB)	Alguns megabytes	Alguns kylobytes, limitado apenas pela memória do celular	Alguns megabytes
Penetração no Mercado	Grande e em crescimento	Grande e em crescimento	Grande nos Estados Unidos e em crescimento	Pequeno e em crescimento
Instalação por download pela rede telefônica	Sim	Possível, mas evitado por causa do tamanho das aplicações	Sim	Possível, mas evitado por causa do tamanho das aplicações
Executa como código-nativo	Não	Sim	Sim	Sim
Linguagem de programação principal suportada	Java	C++	C/C++	C++, VB, C#, VB.NET.
Suporte a outras linguagens de programação	Não	Sim, inclusive Java	Sim, inclusive Java	Sim, inclusive Java

Tabela 1 – Comparação das Principais Características de J2ME / BREW / Symbian / Windows Mobile

3. Framework e o CMF

Neste capitulo procuramos entender o que é um *framework*, porque, e como usá-lo, para em seguida explicarmos o *framework* que estamos portando da plataforma BREW para a plataforma Windows Mobile.

3.1. Uso de *Frameworks*

Para entendermos sobre *frameworks*, devemos procurar entender primeiramente a questão de reuso de software. Quando dizemos que faremos reuso de software, novos ciclos (ou processos) são introduzidos no ciclo de vida de software:

- Desenvolvimento para reuso e,
- Desenvolvimento com reuso.

O processo de Desenvolvimento para reuso é focado em desenvolvimento recursos reusáveis que deverão ser utilizados no processo de desenvolvimento com reuso. No Desenvolvimento com reuso, recursos são procurados e utilizados, se adequados, no desenvolvimento de um software.

Recursos reutilizáveis em orientação-a-objeto podem ser encontrados num espectro contínuo que vai de objetos e classes até padrões de projeto e frameworks orientados a objetos.

A idéia de recursos reutilizados surgiu na década de 1960 quando foram criadas bibliotecas de rotinas (procedures) e funções. Com a introdução do Simula 67, surgiram os objetos, classes e herança surgindo assim as bibliotecas de classes. Só que ambas são focados no reuso de código, que é bom em termos econômicos, mas como o projeto é o principal conteúdo intelectual do software, ele é mais complexo de criar e re-criar do que simplesmente a codificação.

O problema em reusar bibliotecas de classe (ou funções) é que elas não levam a um grande reuso de software, por exemplo, elas permitem apenas reutilizar pequenos pedaços do código, levando aos programadores a fazer grande parte do código. A fome pelo aumento do reuso e do reuso de

padrões de projetos resultaram na criação dos chamados *frameworks* orientado a objetos.

Existem vários conceitos do que sejam *frameworks* OO, mas o melhor deles é que: “um *framework* é um conjunto de classes que incorporam um projeto abstrato para solucionar uma família de problemas relacionados” [23].

Então podemos dizer que um *framework* consiste de um conjunto de classes (não necessariamente abstratas), cujas instâncias colaboram (incorporam), intencionalmente extensível, ex. reusada (projeto abstrato), e que não precisam ser direcionadas para domínio completo de aplicação (uma família de problemas relacionados), ex. permitindo a combinação de *framework*. Adicionalmente, como são expressos em linguagens de programação, fornecem reuso tanto de projeto como código.

Um *framework* particularmente enfatiza aquelas partes do domínio da aplicação que permanecem estáveis e a relações e interações entre essas partes. Vemos que um *framework* é responsável pelo fluxo de controle.

Em sua tese de doutorado Michael Mattson [24] afirma que a principal diferença entre bibliotecas de classes e um *framework* OO é que na primeira somente a aplicação faz chamadas a biblioteca (mão única) enquanto no caso do *framework*, através das técnicas de herança e *binding* dinâmico a uma relação de mão dupla. Podemos explicar melhor usando um exemplo de operação de copia de arquivo.

Imaginemos que desejamos copiar o conteúdo de tipo de arquivo para outro, em uma biblioteca séria necessário que houvesse um método (ou função) que fizesse especificamente a cópia dos tipos de arquivos, enquanto em um *framework*, bastaria que as subclasses que herdam da classe Arquivo (que possui métodos abstratos de escrita e leitura e o método concreto de copia), implementassem seus respectivos métodos de leitura e escrita, deixando a cargo da superclasse o fluxo (ou execução) da cópia. Mattson define o método copia (do exemplo acima) como um método *template* (gabarito) e os métodos abstratos (de escrita e leitura) de *hooks* (ganchos). Um método *template* é um método concreto que contem a máquina de estados da operação a ser realizada e que utiliza *hooks*, que são implementados nas subclasses.

Mattson sugerem que os benefícios da reusabilidade dos *frameworks* vêm de sua extensibilidade e da inversão do controle. Afirma que a extensibilidade dos *frameworks* vem do conjunto de métodos *hooks,* pois possibilitam a futuras aplicações estenderem a interface do *framework*. Isso implica em dizer que esses métodos explicitamente dividem o projeto do *framewok* em interfaces estáveis e comportamento, e partes variáveis que são customizados de acordo com a aplicação a ser desenvolvida. Ele classifica os *framework* OO em três tipos:

- *White-box framework*: que fazem uso pesado de herança e *binding* dinâmico que estão disponíveis em linguagens OO. A extensibilidade de um *framework* caixa branca (*white-box*) é obtida por primeiro herdar das superclasses do *framework* (quase sempre classes abstratas) e em seguida sobrescrever os métodos *hook* pré-definidos. São freqüentemente conhecidos como direcionados-a-arquitetura (*architecture-driven*), ou focados-em-herança(inheritance-focused).
- *Black-box framework*: contam com interfaces definidas de componentes (objetos) e composição de objetos que estão em conformidade com as interfaces. A extensiblidade do *framework black-box* é atingida por primeiro definir componentes que estejam em conformidade com um interface em particular para em seguida integrar esses componentes ao *framework*. Eles são comumente chamados de *data-driven* (direcionados a dados) ou *composition-focused* (focados em composição) *frameworks*.
- *Visual Builder framework*: esse tipo de framework é uma especialização do tipo caixa-preta (*black-box*) são utilizados para alguns domínios de problemas. Uma aplicação é constituída de várias partes. Primeiramente, um script (ou linguagem de script é um tipo de linguagem de programação pouco mais simples, e extremamente direcionadas um domínio especifico.) é usado para unir os componentes e ativá-los. A outra parte é o comportamento dos objetos individualmente, que é provida pela *framework* caixa-preta. Assim, extensão do fram*e*work no caso de caixa-preta é principalmente um script que é similar para todas as aplicações

baseadas no *framework*. Podendo agora utilizar um construtor visual (*visual builder*) que é responsável pela especificação dos objetos a serem incluídos e suas interconexões. Utilizando-se de interface gráficas amigável, especialista no domínio da aplicação podem construir mais rápido e facilmente.

Para desenvolver e utilizar um *framework* é necessário realizar três grandes atividades, analise do domínio do(s) problema(s), projetar (e implementar) o *framework*, e instanciar.

- Análise do Domínio do Problema: Análise do domínio pode ser vista com uma análise ampla e mais extensiva que tenta capturar os requisitos do completo domínio do problema incluído futuros requisitos. Dependendo do problema diferentes fontes de informações são usadas para obter conhecimento sobre o domínio. Exemplos de fontes são experiências anteriores no domínio, especialistas no domínio e padrões existentes.
- Projeto do *Framework*: O objetivo dessa atividade é finalizá-la com um *framework* flexível. Essa atividade é consumidora de esforços, já que é difícil encontrar a abstração correta e identificar partes variáveis e estáveis do *framework*, que freqüentemente resulta em várias iterações de projeto. Para aumentar a *extensibilidade* e flexibilidade do *framework* para atender necessidades futuras das instâncias, padrões de projeto [25] podem ser usados. O *framework* resultante pode ser *white-box*, *black-box*, ou *visual builder* dependendo do domínio, esforço disponível e a intenção dos usuários do *framework*.
- Instanciação do *Framework*: A instanciação de um *framework* difere dependendo do tipo de *framework* (*white-box*, *black-box* ou *visual builder*). Um *framework* pode ser instanciado um ou mais vezes dentro da mesma aplicação ou por aplicações diferentes. A caso mais complexos que a instanciação pode levar a uma aplicação funcional.

Os principais problemas relacionados à utilização de um *framework* podem associados a três atividades acima relacionadas. O primeiro é relativo ao domínio do problema, muitas vezes é difícil analisar certos problemas que podem ter resoluções conflitantes o que implica em decidir qual caminho seguir. A segunda questão é relacionada à própria implementação do *framework*, pois por mais extensível e flexível que se deseje um *framework* ele precisa ser liberado para uso, ou seja, definir um release (liberação). O que impacta em escolher que funcionalidades devem ser sacrificadas para que o *framework* esteja pronto para uso, além de não exagerar no número de funções (recursos) que o *framework* terá isso pode influenciar no desempenho do aplicativo final, levando a desistência do uso do *framework*. E a terceira está associada basicamente à curva de aprendizagem do *framework*, por mais que seus idealizadores achem e certifique que seus frameworks sejam eficientes em relação ao desenvolvimento direto. E altamente que seja feito uma boa documentação do *framework*, principalmente através do uso de exemplos.

3.2. C.E.S.A.R. *Mobile Framework*

O C.E.S.A.R. *Mobile Framework* (CMF) como o próprio nome sugere é um *framework* direcionado para aplicações para dispositivos móveis, ele vem sendo desenvolvido por uma equipe do C.E.S.A.R. para ser utilizando como base para futuras aplicações. Projeto por Tiago Barros, ele tem como objetivo desenvolver um conjunto de padrões de projeto e bibliotecas que poderão facilitar não somente a construção, mas também facilitar a portabilidade de aplicativos entre as diversas plataformas de desenvolvimento para celular (veja capitulo dois para maiores informações). Inicialmente devido à necessidades internas do próprio C.E.S.A.R. o CMF foi desenvolvido em cima da plataforma BREiW em sua versão 2.1.

De maneira a entendermos o CMF vão utilizar a atividades citadas na seção anterior. Desta forma certas decisões poderão ser melhores explicadas.

3.2.1. Análise do Domínio do Problema.

O problema que o CMF propõe-se a resolver está no domínio de aplicações móveis para celulares de baixa a média complexidade que envolva. O CMF procura ser o mais genérico possível de forma a ser base para qualquer tipo de aplicação móvel, facilitando a portabilidade de programas entre plataformas de desenvolvimento para celulares que utilizem a linguagem de programação orientada-a-objetos C++ [26]. Além de melhor o uso de recurso, que são bastante escassos, como memória, acesso a arquivos, rede, etc. E reuso de componentes como por exemplos as telas.

3.2.2 Projeto do *Framework*.

É complicado explicar como foi o projeto do *framework*, pois ele ainda está sendo aperfeiçoado, a cada novo projeto em que ele é utilizado novos recursos são adicionados a ele. A intenção original de Tiago Barros é criar um *framework* do tipo caixa-preta (*black-box*) onde os mecanismos de controle da máquina de estados e gerenciamento de telas (e gráficos) ficam dentro do *framework*, cabendo o usuário criar classes que representam os estados (e telas). Depois de criar as classes essas são dinâmicas adicionadas os gerenciadores (estado e telas) que controlaram a mudança dos estados (ou telas), mas quem determinar a mudança entre os estados é o próprio usuário, (ex.: através de respostas a eventos).

O *framework* é atualmente constituído de quatro grupos de componentes (algo semelhantes a pacotes) básicos (*base*, *system*, *screen*, *resources*), quatro "*plugins*" (*addressbook*, *file*, *timer*, *network*) e um exemplo (*hello*). Vamos explicar cada componente, parte dessas informações encontra-se em [27]:

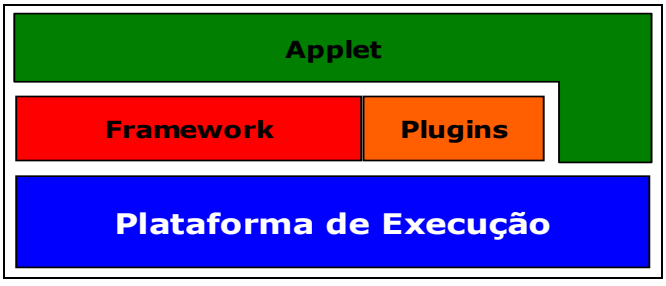

Figura 5. Visão da Arquitetura do CMF.

3.2.2.1 *Base*

Onde ficam agrupadas as classes que compõem o *core* do *framework*. Responsável pela abstração da inicialização em relação à plataforma que está sendo executada. Também conta com as classes básicas que fornecem serviço às demais classes. É composto das seguintes classes:

- *CMF_CApplication* : Responsável pela inicialiazação do *framework*, e aplicativos associado;
- *CMF_CIterator*: Classe base para todos os iteradores;
- *CMF_CObject*: Classe base *object*, que no caso BREW, sobrescreve os operadores ***new*** e ***delete*** para um comportamento próximo as classes *Object* de outras versões de C++.
- *CMF_CObjectManager*: Classe *template* que serve de gerenciar objetos (ou o tipo utilizado na herança);
- *CMF_CStateMachine*: Classe responsável controla o fluxo entre os estados;
- *CMF_MApplet*: Classe que representação a aplicação do usuário;
- *CMF_MState*: Classe que representa um estado do programa.
- *CMF_TEventID*: Arquivo de cabeçalho usado apenas para definir os valores de alguns eventos.

3.2.2.2 *System*

O componente é composto de somente duas classes: *CMF_CSystem* que permite criar outros objetos de sistema, conforme tipo informado e a *CMF_MPlugin* que serve de base para todos os *plugins*.

3.2.2.3 *Screen*

Neste componente ficam todas as classes responsáveis pela manipulação de gráficos e telas do *framework*. É formado pelo gerenciador de telas (*CMF_CScreenManager*) que tem características semelhantes à classe *CMF_CStateMachine* e responsável de fluxo de controle da telas. As telas são dividias em duas categorias *Canvas* e Diálogos. Na primeira é criada uma área livre para desenho e manipulação de gráfico, já a segunda é destinada a construção de telas que já possuem componentes embutidos

(como *softkeys* – teclas de seleção em celulares, e título). As demais classes dão suporte a esse dois tipos de telas. E são basicamente componentes de tela (menu, texto, campo de texto, etc.) e manipulação de gráficos (linhas, *bitmaps*, etc.).

3.2.2.4 *Resources*

O último dos pacotes é responsável pelo controle de acesso aos arquivos de recursos. Esses arquivos de recursos no caso de BREW (veja seção 2.2.1) são colocados em arquivos separados, e nele ficam três tipos de elementos: *strings*, diálogos e imagens.

3.2.2.5 Plugins

Os quatros plugins desenvolvimento juntamente com a primeira versão do *framework* são: AddressBook, que fornece acesso a Agenda Telefônica do celular; Timer, de fornecer serviço de temporização; File, para manipulação de arquivos; e Network, para acesso a rede via *sockets* e protocolo HTTP.

3.2.2.6 Aplicação Hello

Essa aplicação é um teste para o *framework*, serve de exemplo para que usuários que queriam utilizar o *framework*, tenham uma referência. O nome foi uma homenagem à famosa mensagem utiliza por quem está aprendendo uma linguagem de programação: "*Hello World!*". A aplicação faz uso de todos os *plugins* implementados realizado testes em cada um deles.

3.2.3 Instanciação do *Framework*

Nesta subseção veremos como um usuário deve utilizar o *framework* para criar uma aplicação. O material a seguir está mais bem detalhado na seção "Construindo uma aplicação – Ações:" em [27].

1. Definir os estados contemplados pela aplicação;
2. Definir os eventos do usuário que a aplicação ira tratar;

3. Criar arquivos de recursos (strings, diálogos, e imagens), já pensando nos idiomas a serem utilizados pela aplicação (ex.: português, inglês, espanhol);

4. Para cada estado da aplicação, deverá ser construída uma classe para representá-lo; Caso sejam verificados características ou comportamentos em comuns entre (todos) os estados, deve ser criada uma classe especifica (*CMF_CStateAny*) para encapsular essas características / comportamento comuns.

4.a. Herda-se da classe base (*CMF_MState*) ou (*CMF_CStateAny*);

4.b. Implementam-se os métodos abstratos (OnEnter, OnResume, OnExit). Em caso de eventos definidos no passo 2, criar os métodos específicos a cada evento dentro da classe.

4.c. E no construtor da classe é necessário: adicionar o estado associado à classe na máquina de estado; realizar alocações de memória no método *ConstructorL*; utilizar o método *MapEvent* para fazer o mapeamento dos eventos; e quaisquer outras inicializações necessárias.

5. Para cada diálogo da aplicação, seguir passos parecidos como os estados: Criar classe e herdar de classe base, Implementar métodos abstratos, e realizar inicializações necessárias.

6. Modificar o arquivo BID (BREW Class ID), adicionando o nome da classe que implementa a aplicação (o applet).

7. Implementar a classe que conterá a aplicação. Essa classe deverá:

7.a. Definir o número máximo de diálogos;

7.b. Herdar da classe *CMF_MApplet*;

7.c. Implementar os métodos abstratos;

7.d. Implementar Construtor e Destrutor;

7.e. Ao implementar o método *ConstructL*, o usuário deverá fazer: uma chamada aos métodos *ConstructL* das classes *CMF_CStateMachine* e *CMF_CScreenManager*; Instanciar diálogos e estados; Se necessário configurar as softkeys.

7.f. Implementar o método *GetLanguage*: que deve retornar uma estrutura (*CMF_TResFileSet*) contendo os nomes dos arquivos de recursos a serem utilizados pela aplicação;

4. Windows Mobile™

Neste capitulo veremos a história do Microsoft *Windows Mobile*, passando por cada uma de suas versões. Seguiremos falando sobre como está estruturado o *Windows Mobile*, seus componentes, sua API. Na parte final faremos uma breve avaliação sobre a tecnologia *Windows Mobile*.

4.1. História e Versões

Falarmos de história do *Windows Mobile* é voltarmos para ano de 2002, quando a Microsoft tomou uma decisão estratégica de criar uma nova marca, uma nova plataforma para atingir um mercado de *PDA*s e *Smartphones*. A intenção da Microsoft era criar um padrão de hardware mínimo, podendo estabelecer assim uma API comum, garantindo que o mesmo código fosse utilizando em equipamentos de fabricantes diferentes. O *Windows Mobile* é baseado no sistema operacional *Windows CE .NET*. Em 2003 a Microsoft fez o lançamento oficial do *Windows Mobile* 2003[28], levando uma série de novidades para o mundo dos equipamentos móveis, entre as principais novidades:

- Maior conectividade: Configuração Zero, *Bluetooth*, 802.1x, *IPv6*, Múltiplos *VPNs*;
- Maior Segurança: Gerenciador de Certificado, *IPSEC*, Criptografia 128-bit;
- Multimídia: Windows Media Player 9, Movie Maker 2;
- Desenvolvimento: *.NET Compact Framework*, Melhor suporte a desenvolvimento;

Em 2004 a Microsoft lança a versão *Windows Mobile 2003 Second Edition*[29], que traz poucas novidades: suporte a rotação da tela, nova resoluções de telas, melhoria no *ActiveSync* (ferramenta de sincronização do equipamento móvel com o PC) e o *.NET Compact Framework* já embutido no ROM do equipamento. Se novas alterações nas API para aplicações. Em maio deste ano a Microsoft lança o Windows Mobile 5[30]. que traz grandes novidades como: Novas *API* para desenvolvimento de

aplicações que utilizem GPS, Gerenciamento de Imagens, Gerenciamento de Contatos, Maior portabilidade de código entre as diferentes edições, Novo sistema de notificação de chamada e SMS, Desenvolvimento utilizando o *Visual Studio 2005*, entre vários outras.

A Microsoft trabalha com três edições do *Windows Mobile*:

- *Windows Mobile for Pocket PC Edition*
- *Windows Mobile for Pocket PC Phone Edition*
- *Windows Mobile for Smartphone Edition*

Como sabemos a existem diferenças entre essas edições, utilizemos a versão *Pocket PC Edition* como referência, A versão *Pocket PC Phone Edition* traz somente a capacidade de gerar e receber ligações telefônicas, já na versão Smartphone Edition que também pode receber e gerar ligações, mas infelizmente não possui recursos como versões móveis do Outlook, Word e Excel, conectividade Bluetooth e Wi-Fi. Os dois últimos itens podem ser fornecidos pelo fabricante do aparelho.

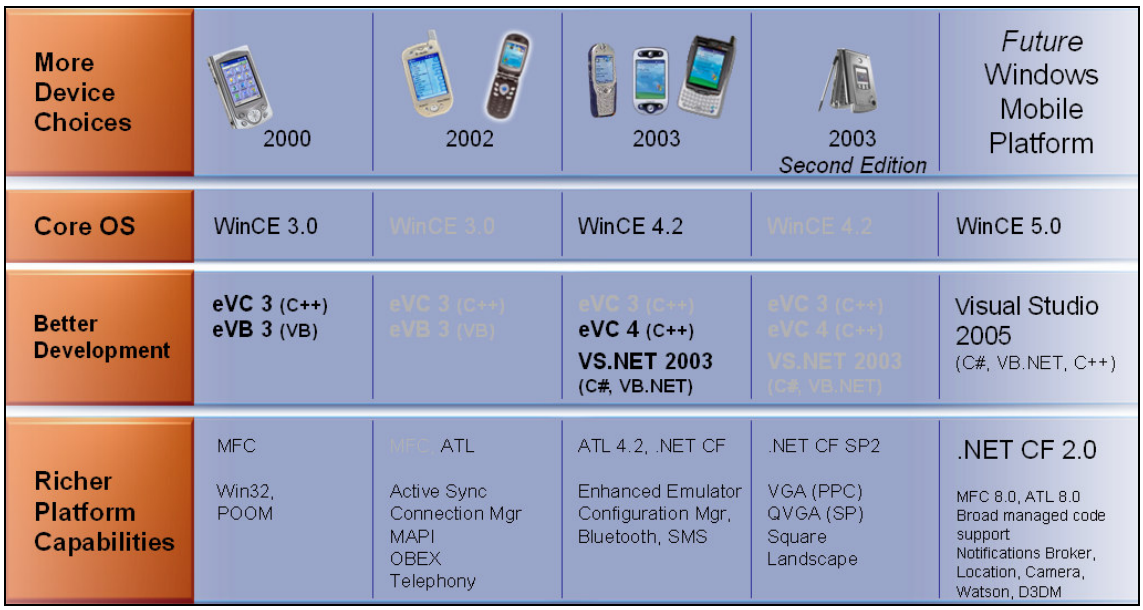

Figura 6. Evolução do *Windows Mobile*

4.2. Estrutura

A estrutura do *Windows Mobile* 2003 é apresentada na figura 7. Nela notamos a presença de várias camadas. As camadas inferiores (em tons de cinzas) ficam a critério dos fabricantes de hardware, de um lado está a parte relativa ao processamento, gerenciamento de memória, tela, e comunicação com o PC (via *USB* ou Serial), que devem atender os padrões mínimos da Microsoft. Do outro lado é hardware responsável pela comunicação sem-fio (opcional), permitindo que várias tecnologias sejam agregadas ao equipamento. Logo acima temos a cada de *Drivers* que fazem a interface com o sistema operacional, normalmente esses *Drivers* são desenvolvidos pelos próprios fabricantes de hardware, ou podem já está disponível pela Microsoft ou terceiros.

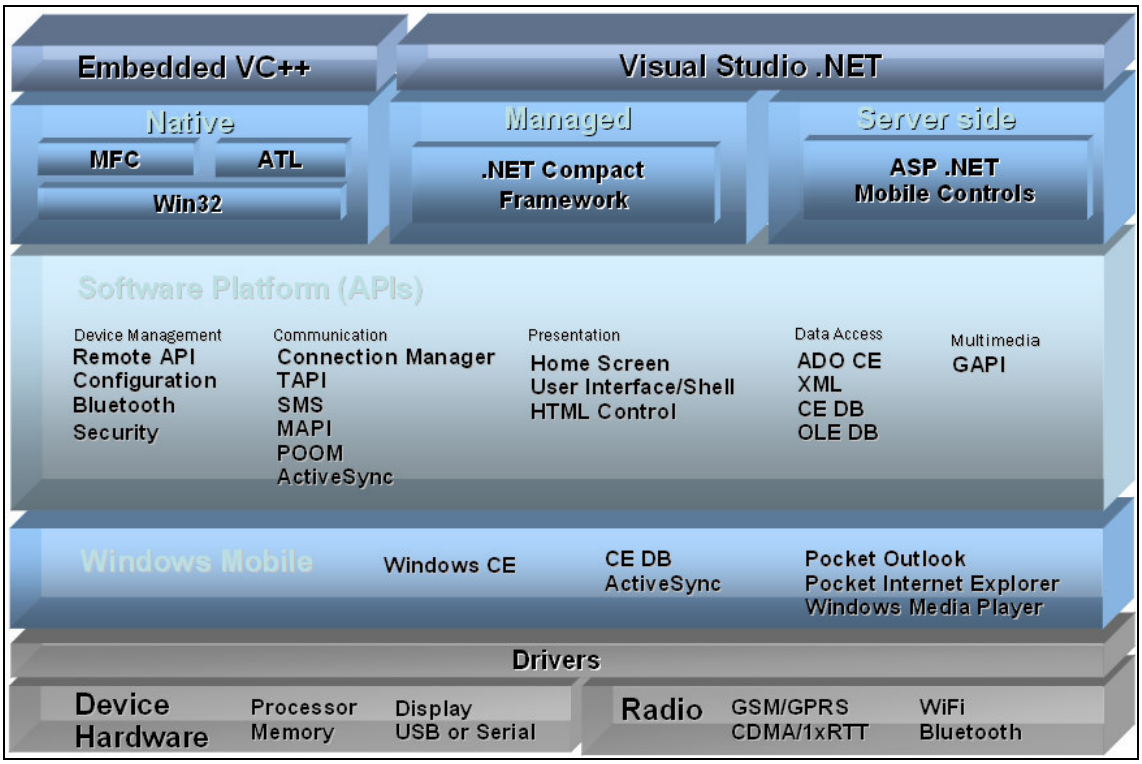

Figura 7. Estrutura do *Windows Mobile 2003*.

Logo acima está o sistema operacional Windows CE, e serviços associados: acesso à banco de dados (CE DB), e sincronização; e os aplicativos *Pocket Outlook*, *Internet Explorer* e "*Windows Media Player*". Esse conjunto (sistema operacional, serviços e programas) forma, juntamente com o hardware a plataforma *Windows Mobile*.

A Microsoft construiu um extenso conjunto de *API*s para auxiliar no desenvolvimento de software e dividiu em algumas categorias: Gerenciamento do Dispositivo, Comunicação, Apresentação, Acesso a dados, e Multimídia. (representada pela camada azul clara).

Acima das API estão os chamados *frameworks* (alguns são apenas APIs) da Microsoft. Como vemos estão divididos em três grupos o Nativo, Gerenciável, e Servidor. O Nativo agrupa a API *Win32* (comum a todos o sistemas operacionais da família Windows), a *MFC* (*Microsoft Foundation Classes*) que é um conjunto de classes e objetos, que facilitam a vida da programação orientada a objeto, e *ATL*(Active Template Library) é um conjunto de classe de gabaritos C++ (*templates*) utilizado para construção de componentes COM (*Common Object Model*). O Gerenciável contém o chamado *.NET Compact Framework* [31] uma framework desenvolvimento pela Microsoft em 2003 para concorrer com a plataforma Java, suas principais características são: código portável, utilização de múltiplas linguagens, *XML web services*, rica e poderosa biblioteca de classes, e código gerenciável (de forma a assegurar melhor uso de memória e recursos). O que aqui identificamos como Servidor, nada mas é que somente componentes do lado do servidor direcionados para uso de clientes móveis.

E finalmente no topo dessa estrutura estão os ambientes de desenvolvimento (ou ferramentas) que veremos na próxima seção.

4.3. Ferramentas

Sempre que formos desenvolver uma aplicação precisamos utilizar ferramentas, com editores de textos e compiladores. Hoje em dias as *IDE* (Ambientes Integrados de Desenvolvimento) unifica uma série de ferramentas é um único programa. Quando formos trabalhar com *Windows Mobile* [32], existem duas opções de desenvolvimento. Quem for trabalhar com código Nativo (*API Win32* e *MFC*) tem a opção de utilizar o Microsoft *embedded Visual C++* (figura 8) uma versão simplifica e direcionada o Microsoft *Visual C++* 6.0, já quem deseja

trabalhar em cima do *.NET Compact Framework* deve utilizar o Microsoft Visual Studio .NET 2003 (figura 9).

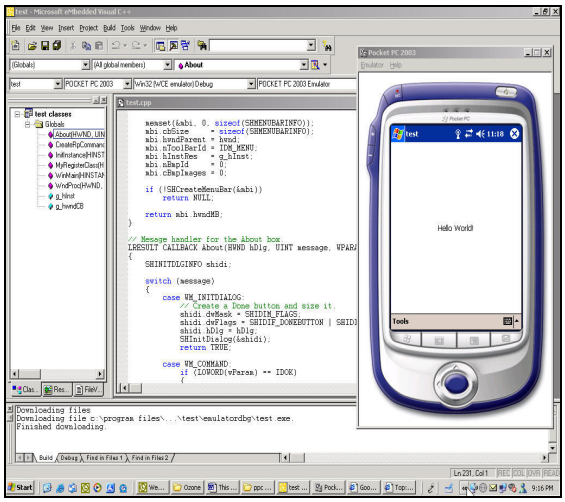

Figura 8. embedded Visual C++.

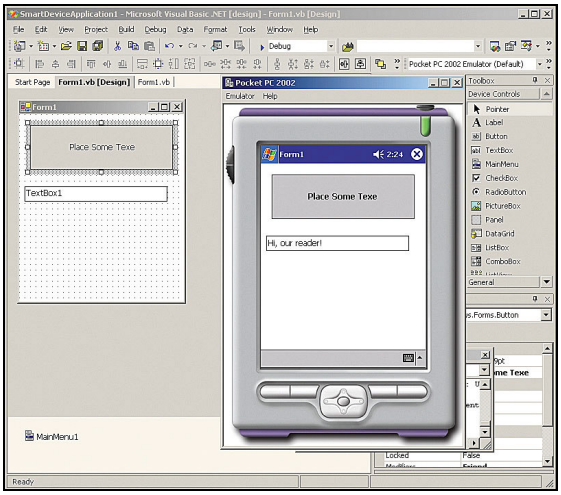

Figura 9. Visual Studio .NET 2003

Em ambos os casos devem ser instalados também um kit de desenvolvimento (*SDK*), existe kit para o *Pocket PC* e outro para *Smartphones*. No kit está incluído emuladores, bibliotecas e mini-ferramentas que auxiliam o desenvolvimento de aplicativos para as respectivas edições. De maneira a tornar a emulação dos programas nos dispositivos ainda melhor, a Microsoft impôs o uso do *ActiveSync*, que faz a sincronização de dados e transferência de arquivos entre o dispositivo móvel e o PC. Fora as novidades dentro do dispositivo móvel a versão 5 da plataforma *Windows Mobile* traz novidades fora dela, como a utilização do *Visual Studio .NET 2005* [33] (ainda em beta) que

unifica em um sistema o desenvolvimento nativo e gerenciável (*.NET Compact Framework*).

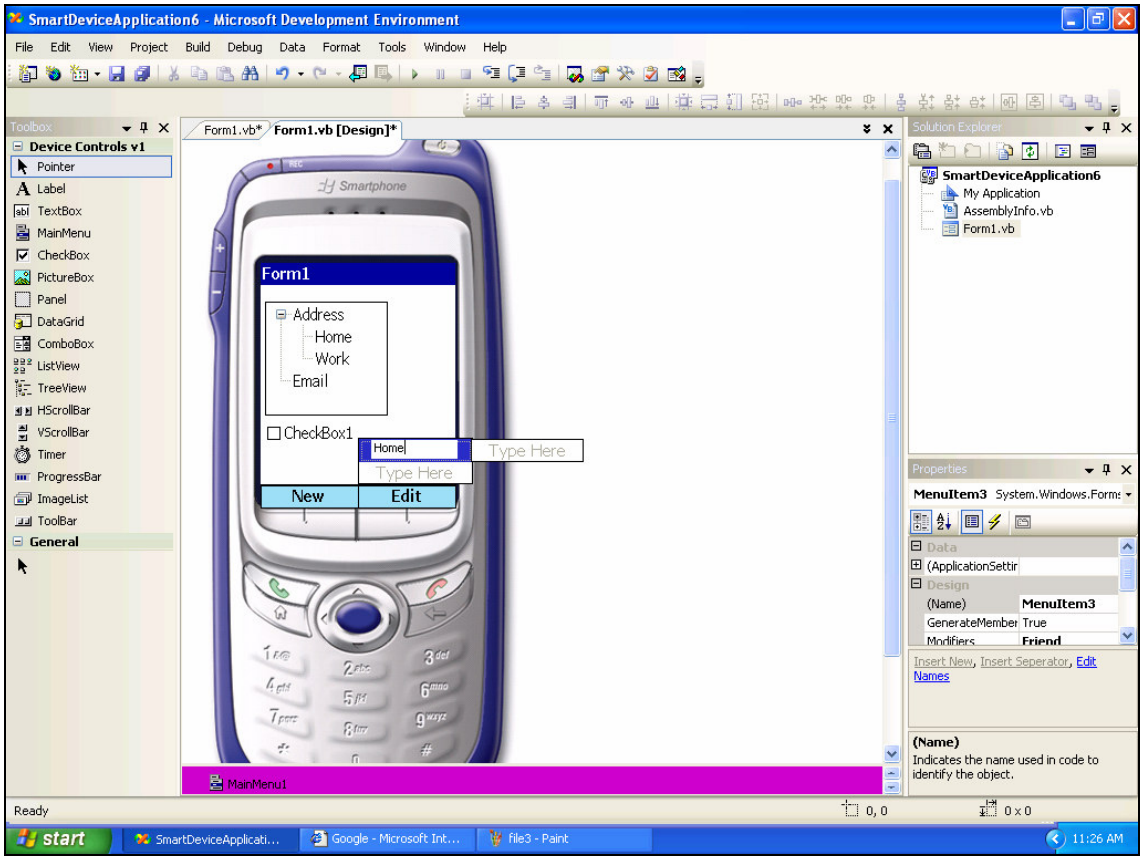

Figura 10. Visual Studio 2005

4.4. Desenvolvimento

O desenvolvimento de um aplicativo para dispositivos móveis que usam a plataforma *Windows Mobile* é semelhante ao desenvolvimento de aplicativos para qualquer outro sistema operacional da família *Windows* já que tem como base o sistema operacional *Windows CE*. E assim como os outros *Windows* utiliza um modelo de programação a evento (*event-driven*)[34]. Um programa *Windows CE* recebe mensagens, interpreta as mensagens, e age de acordo com as mensagens.

Um programa *Windows CE* tem um ou mais janelas, que recebem e processam mensagens num *loop*. A janelas podem ser visíveis ou não. Cada janela possui um manipulador(*handle*) de janela (hWnd) juntamente com um processador de mensagem que cuida da mensagens da janela.

Pode-se também usar o manipulador de janela para chamar qualquer função relacionada.

Como em outros plataformas Windows, um programa Windows CE possui duas funções primárias, um processador de mensagem (normalmente chamado de *WndProc*) e *WndMain*, que fornece um ponto de entrada para a aplicação. A função *WndProc* processa as mensagens para a Janela do programa. Em geral, uma aplicação processa mensagens destinadas a ela, repassando o restante para o sistema operacional. Além de ser o processador primário de mensagens de uma aplicação, *WinMain* trata da inicialização e finalização (veja exemplo na figura 11).

Quando for desenvolver para o *Windows Mobile* (*Windows CE*) deve-se ter o cuidado de escolher qual o hardware (*Smartphone* ou *Pocket PC*) e processador (normalmente da família *ARM*[35]). Isso é determinado principalmente pelo SDK (Kit de Desenvolvimento de Software) utilizado. É sempre bom verificar junto ao fabricante do dispositivo quaisquer particularidades do mesmo.

E como vimos na seção anterior é permitido ao desenvolvedor escolher entre uma série de ferramentas, como programar usando diretamente a *API Win32* ou ferramentas (e framework) mais avançadas como *MFC* ou *ATL*.

As principais diferenças de programação entre o *Windows Mobile* e outras plataformas Windows, pode ser resumidos a quatros pontos [36]:

- Poucos Recursos: Baixa resolução; Pouco espaço para armazenamento de dados (e programas), pouco dispositivos contam com disco rígido; Não existe memória virtual; e pior pouca memória RAM para execução do Programa.
- Unicode: *O Windows Mobile* adota a utilização do padrão *Unicode*, em que cada caractere é representando por dois bytes, levando ao programador a ficar mais atento na hora de alocar um *buffer* para texto usando esse formato. Para isso foi criado um tipo *TCHAR* que abstrai o uso do *Unicode* e do *ASCII* (de 8-bits).
- Novos controles: Como possuir recursos limitados, cada espaço é utilizado para levar funcionalidade para o usuário final. Desta maneira novos controles, como por exemplo uma barra de menu-

status que unifica as funcionalidades de ambos e fazendo ocupar menos espaço. E alguns componentes que estão disponíveis para as plataformas *Windows* para PC foram removidos.

- Componentização: Apesar do esforço da Microsoft para aproximar *Pocket PC* dos *Smartphones* certas *API* que são especificas para cada edição do *Windows Mobile*. O que implica em dizer que certos ajustes devem ser feitos para que uma mesma aplicação rode em ambos os sistemas.
- Subconjunto do *Win32*: Apesar ser dito que o *Windows Mobile* possui um *API Win32*, na verdade a plataforma possui somente um subconjunto da *API Win32* que é usado em PCs. Dificultando a portabilidade de programa de PC para *Pocket* e *Smartphone*.

```
// HelloWorld.cpp : Defines the entry point for the application.
//

#include <windows.h>
#include <windowsx.h>
#include <aygshell.h>
#include "resource.h"

HINSTANCE g_hInst = NULL;  // Local copy of hInstance

#define ARRAYSIZE(a)   (sizeof(a)/sizeof(*a))

const TCHAR* g_szAppWndClass = TEXT("HelloApp");

TCHAR g_szMessage[30];

/**********************************************************************
   OnCreate

**********************************************************************/
LRESULT OnCreate(
    HWND hwnd,
    CREATESTRUCT* lParam
    )
{
    // create the menu bar
    SHMENUBARINFO mbi;
    ZeroMemory(&mbi, sizeof(SHMENUBARINFO));
    mbi.cbSize = sizeof(SHMENUBARINFO);
    mbi.hwndParent = hwnd;
    mbi.nToolBarId = IDR_HELLO_MENUBAR;
    mbi.hInstRes = g_hInst;
    if(!SHCreateMenuBar(&mbi))
    {
        // Couldn't create the menu bar.  Fail creation of the window.
        return(-1);
    }

    // Get our message text.
```

```
    if(0 == LoadString(g_hInst, IDS_HELLO_MESSAGE, g_szMessage,
ARRAYSIZE(g_szMessage)))
    {
        // Couldn't load the string.  Fail creation of the window.
        return(-1);
    }

    // Do other window creation related things here.

    return(0); // continue creation of the window
}

/*********************************************************************
   WndProc
*********************************************************************/
LRESULT CALLBACK WndProc(
    HWND hwnd,
    UINT msg,
    WPARAM wp,
    LPARAM lp
    )
{
    LRESULT lResult = TRUE;

    switch(msg)
    {
        case WM_CREATE:
            lResult = OnCreate(hwnd, (CREATESTRUCT*)lp);
            break;

        case WM_COMMAND:
            switch (wp)
            {
                case IDOK:
                    DestroyWindow(hwnd);
                    break;
                default:
                    goto DoDefault;
            }
            break;

        case WM_PAINT:
            {
                HDC hdc;
                PAINTSTRUCT ps;
                RECT rect;

                hdc = BeginPaint(hwnd, &ps);
                GetClientRect(hwnd, &rect);

                DrawText(hdc, g_szMessage, -1, &rect, DT_SINGLELINE |
DT_CENTER | DT_VCENTER);
                EndPaint (hwnd, &ps);
            }
            break;

        case WM_DESTROY:
            PostQuitMessage(0);
            break;

        DoDefault:
```

```
        default:
            lResult = DefWindowProc(hwnd, msg, wp, lp);
            break;
    }

    return(lResult);
}

/**********************************************************************
   ActivatePreviousInstance
**********************************************************************/
HRESULT ActivatePreviousInstance(
    const TCHAR* pszClass,
    const TCHAR* pszTitle,
    BOOL* pfActivated
    )
{
    HRESULT hr = S_OK;
    int cTries;
    HANDLE hMutex = NULL;

    *pfActivated = FALSE;
    cTries = 5;
    while(cTries > 0)
    {
        hMutex = CreateMutex(NULL, FALSE, pszClass); // NOTE: We don't
want to own the object.
        if(NULL == hMutex)
        {
            // Something bad happened, fail.
            hr = E_FAIL;
            goto Exit;
        }

        if(GetLastError() == ERROR_ALREADY_EXISTS)
        {
            HWND hwnd;

            CloseHandle(hMutex);
            hMutex = NULL;

            // There is already an instance of this app
            // running.  Try to bring it to the foreground.

            hwnd = FindWindow(pszClass, pszTitle);
            if(NULL == hwnd)
            {
                // It's possible that the other window is in the
process of being created...
                Sleep(500);
                hwnd = FindWindow(pszClass, pszTitle);
            }

            if(NULL != hwnd)
            {
                // Set the previous instance as the foreground window

                // The "| 0x01" in the code below activates
                // the correct owned window of the
                // previous instance's main window.
                SetForegroundWindow((HWND) (((ULONG) hwnd) | 0x01));
```

```
                // We are done.
                *pfActivated = TRUE;
                break;
            }

            // It's possible that the instance we found isn't coming
up,
            // but rather is going down.  Try again.
            cTries--;
        }
        else
        {
            // We were the first one to create the mutex
            // so that makes us the main instance.  'leak'
            // the mutex in this function so it gets cleaned
            // up by the OS when this instance exits.
            break;
        }
    }

    if(cTries <= 0)
    {
        // Someone else owns the mutex but we cannot find
        // their main window to activate.
        hr = E_FAIL;
        goto Exit;
    }

Exit:
    return(hr);
}


/*****************************************************************
  WinMain
*****************************************************************/

int WINAPI WinMain(
    HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPWSTR lpCmdLine,
    int nCmdShow
    )
{
    MSG msg;
    HWND hwnd = NULL;
    BOOL fActivated;
    WNDCLASS wc;
    HWND hwndMain;
    TCHAR szAppTitle[20];

    g_hInst = hInstance;

    if(0 == LoadString(g_hInst, IDS_HELLO_TITLE, szAppTitle,
ARRAYSIZE(szAppTitle)))
    {
        return(0);
    }
```

```
    if(FAILED(ActivatePreviousInstance(g_szAppWndClass, szAppTitle,
&fActivated)) ||
            fActivated)
    {
        return(0);
    }

    // Register our main window's class.
    ZeroMemory(&wc, sizeof(wc));
    wc.style = CS_HREDRAW | CS_VREDRAW ;
    wc.lpfnWndProc = (WNDPROC)WndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hIcon = NULL;
    wc.hInstance = g_hInst;
    wc.hCursor = NULL;
    wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName = NULL;
    wc.lpszClassName = g_szAppWndClass;
    if(!RegisterClass(&wc))
    {
        return(0);
    }

    // Create the main window.
    hwndMain = CreateWindow(g_szAppWndClass, szAppTitle,
            WS_CLIPCHILDREN, // Setting this to 0 gives a default
style we don't want.  Use a benign style bit instead.
            CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
CW_USEDEFAULT,
            NULL, NULL, g_hInst, NULL );
    if(!hwndMain)
    {
        return(0);
    }

    ShowWindow(hwndMain, nCmdShow);
    UpdateWindow(hwndMain);

    // Pump messages until a PostQuitMessage.
    while(GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg);
        DispatchMessage(&msg);
    }
    return msg.wParam;
}

// end HelloWorld.cpp
```

Figura 11. Exemplo de um programa *Windows Mobile for Smartphone*.

4.5. Conclusões

Vimos como surgiu e como evoluiu o *Windows Mobile*. Sua arquitetura e seus principais componentes. Mas com tantas versões e com a falta de

recursos (cronológicos, financeiros, e humanos) precisamos tomar uma decisão sobre qual versão do *Windows Mobile* nós iremos utilizar como base para o estudo do *porting* do *framework*. Baseando-se nas informações apresentadas acima a versão *Windows Mobile 2003 (Second Edition) for Smartphone* se torna a mais apropriada, por dois motivos: Ser a primeira a der uma versão dedicada para a telefonia (*smartphones*), e ter um bom suporte da Microsoft e do Mercado. A versão 5.0 ficou de fora porque é recente (lançada em julho de 2005) e da necessidade de se utilizar o Microsoft *Visual Studio* 2005 (que ainda não foi lançada oficialmente e possui mais de um gigabyte de tamanho).

5. Adaptando CMF para *Windows Mobile* 2003

Neste capitulo iremos expor o estudo sobre a portabilidade do C.E.S.A.R. *Mobile Framework*, que hoje se encontra implementando parcialmente na plataforma *BREW*, para a plataforma para celulares da Microsoft, o *Windows Mobile 2003 for Smartphone*. Esse estudo tem como metodologia a analise do projeto atual (em *BREW*), levantando a funcionalidade de cada parte para somente assim sugerir uma solução para que essa funcionalidade permaneça, sem grandes mudanças quando em nova plataforma (*Windows Mobile*).

5.1. *BREW* vs. *Windows Mobile 2003*

É de conhecimento geral que todas as plataformas de desenvolvimento tem suas diferenças e algumas possuem princípios parecidos, mas infelizmente não o suficiente para que um aplicativo possa ser fácil, rápido e corretamente transportado entres essas plataformas. Não fugindo dessa regra BREW e *Windows Mobile* apresentam grandes diferenças e poucas semelhanças.

Precisamos levantar as diferenças e semelhanças que possam impactar , seja de forma positiva, seja de forma negativa. É essencial que possamos ter esses dados, pois decisões a serem tomadas sobre as soluções a serem sugeridas para a permanência das funcionalidades não sejam feitas de forma aleatória ou arbitrária.

Durante esse estudo os seguintes pontos sobre as principais e mais relevantes diferenças e também semelhanças foram descobertos:

- Linguagem de programação: ambas as plataformas utilizam-se das linguagens C e C++ para fazer os aplicativos, mesmo que nenhuma das duas sigam os padrões existentes [37 e 38] . Mas de forma a torna mais homogenia utilizaremos apenas a linguagem C++, pois como visto no capitulo três, os *frameworks* são comuns por usarem o paradigma orientado a objeto do que o imperativo (o que não implica em dizer que não existam *frameworks* imperativos ou funcionais).

- Uso de IDEs Microsoft para o desenvolvimento da solução: Em ambas plataformas é necessária à utilização de soluções Microsoft para desenvolvimento de software, em *BREW* é utilizado o *Visual C++* 6.0 enquanto o *Windows Mobile 2003*, quando programa-se nativamente, utiliza o *Embedded Visual C++ 4.0*. Ambas as ferramentas compartilham componentes e funcionalidades em comum, pois foram construídos sobre uma plataforma comum.
- Sistemas de Recursos: um ponto negativo, e extremamente complicado de solucionar é o sistema de recursos (diálogos, formulários, telas, textos, imagens, etc.) utilizado por *BREW* e o *Windows Mobile*, enquanto na primeira é utilizada uma ferramenta a parte, gerando um arquivo separado, o segundo embuti no arquivo executável os recursos a serem utilizado pela aplicação, apesar de usarem o sistema de Identificadores para acessar esses recursos, utilizam metodologia diferente em relação ao carregamento e manipulação dos mesmos.
- *API*: as duas plataformas apresentam um conjunto de *APIs* que procuram atender todas as necessidades dos usuários (mesmo sabendo que isso é impossível). Mas essa variedade de funcionalidade apresenta uma diversidade gigantesca na forma de utilização desses conjuntos de funções. Na plataforma *BREW* as *APIs* são instanciadas como se fossem programas. No Windows *Mobile* as *API* não devem ser vistas como solução para as deficiências de *BREW*, por querer ajudar na migração de *programadores* de PC para o mundo móvel a Microsoft optou por construir as APIs de forma a serem compatíveis (não completamente) em ambos as plataformas (PC e *Mobile*), na forma da famosa *API Win32*. No caso do *Mobile* é estabelecido um subconjunto dela, já que nem todas as funções são necessárias. O problema da API *Win32* é que complexa e foi pensada para programadores C. O *Windows Mobile* conta também com uma biblioteca / *framework* orientada a objeto, a *Microsoft Foundation Classes* (MFC).
- Concorrência e elementos Estáticos: muitos vêm como desvantagem para a plataforma *BREW* a ausência de suporte para concorrência

(apesar de poder ser feita de forma trabalhosa) e de variáveis estáticas (que por outro lado garante maior estabilidade aos programas). O *Windows Mobile* não sofre com isso, nele é possível utilizar variáveis globais e manipulação de *threads* e processos.

Esses são somente alguns dos aspectos que diferenciam essas plataformas e que de algum forma do influenciam as decisões de projeto a serem tomadas quanto a *porting* ou não do CMF.

5.2. Estudo de Soluções

Durante esse estudo de como deveria ser feito o *porting* do *framework*, várias técnicas surgiram para resolver esse problema. Mas somente duas chamaram atenção: Adaptar as Funcionalidades do *Framework*, e Adaptar o Código do *Framework*. Cada solução possui determinadas vantagens e desvantagens. Veremos nas próximas seções o que deve ser feito no framework em cada uma das técnicas acima citadas, expondo as vantagens e desvantagens dessas soluções. De forma a facilitar a compreensão e também o trabalho realizado, apenas partes essências do CMF foram utilizadas.

5.3. Adaptando o Código do *Framework*

Na solução de adaptar o código seja talvez a primeira vista a solução mais simples e rápida de realizar. Mas depois que analisarmos a CMF para verificar que partes do código deverão ser modificadas, alguns acharão que não é tão simples e rápido assim. Nessa solução todas as interfaces, chamadas de métodos, variáveis públicas, e estruturas que possam ser acessadas pelos usuários devem permanecer intactas de forma que a uma aplicação criada na plataforma *BREW* possua ser transportada sem custo nenhum para o usuário, bastando apenas recompilar código na plataforma *Windows Mobile*.

O problema detectado para essa solução está exatamente nessa não-alteração de métodos, variáveis, estruturas. Para que os elementos permaneçam os mesmo é necessária a criação de *wrappers*, que são métodos ou classes que iriam encapsular e "traduzir" as chamadas dos métodos utilizados pelo usuário para o formato da plataforma *Windows Mobile*. O Custo de criar *wrappers* é muito grande devido a quantidade de métodos utilizados pelo CMF, e de quantidades de novas chamadas de funções, o que implicaria numa sobrecarga de processamento e uso da pilha (memória), que em alguns dispositivos móveis é bastante reduzida.

Isso ocorre principalmente devido a uso de elementos *BREW*, como, por exemplo, estruturas e constantes. Uma alternativa talvez fosse a implementação da plataforma *BREW* sobre a plataforma *Windows Mobile*, já que de certo ponto de vista a plataforma Windows é mais completa por se tratar de um sistema operacional. Mas fugiria do escopo do *framework* que é de criar uma camada de software para agilizar a portabilidade e criação de aplicativos em várias plataformas de desenvolvimento.

5.4. Adaptando as Funcionalidades do *Framework*

Nesta solução é sugerido que as características principais do *framework*, como a utilização de uma classe máquina de estados, gerenciador de telas, a aplicação do usuário em formado de *Applet*, ou seja, a Idéia fundamental do *framework* permanece-se modificando ou até mesmo construído do zero, esse modelo de projeto.

O benefício principal é permitir que o framework utilize as melhores funcionalidades de cada sistema, mantendo a idéia de um sistema que diminui os esforços de criar uma nova aplicação, garantindo maior estabilidade e controle na execução da mesma.

O Trabalho para adaptar o *framework* seguindo essa solução é enorme, pois implica em refazer duas etapas da criação de software: o design e a codificação. Além de forçar o usuário a reaprender a utilizar o *framework*. E sem falar de um dos fundamentos do *framework*, o de trazer maior portabilidade para aplicativos projetados para dispositivos móveis celular.

5.5. Viabilidade

Após longas análises das duas soluções, que pareciam ser as mais promissoras, e de uma série de tentativas frustradas de codificar a adaptação do C.E.S.A.R. *Mobile Framework* da plataforma QUALCOMM BREW para a Microsoft Windows Mobile 2003 for Smartphone. É cabível e compressível afirmar que com atual arquitetura, apesar de funcional na plataforma BREW, o *porting* para a plataforma *Windows Mobile* é inviável tecnicamente, pois em decisões feitas em ambas as soluções fundamentos essenciais ao *framework* seriam perdidas.

6. Conclusão

Esse trabalho mostrou que nem as melhores soluções por mais atraentes que pareçam, por mais que suas vantagens que tragam, não são admissíveis se tivermos que perder características que fazem um aplicativo, no caso um *framework* de software para dispositivos móveis, o que ele é.

Ver que na ciência e engenharia tentativas e erros também fazem parte do aprendizado, mas mais importante é que essas lições aprendidas seja utilizadas como experiências para que numa situação parecida que venha acontecer possamos escolher o caminho correto ou pelo menos evitar o mesmo caminho.

Podemos ter uma melhor compreensão das plataformas de desenvolvimento de software para dispositivos móveis em especial celulares, notando que apesar de grandes limitações de recursos como processamento e memória são capazes de realizar grandes tarefas, e com ajuda de *frameworks* como o CMF, facilitar a vida de programadores em suas tarefas de criar e portar aplicações entre várias plataformas.

6.1. Dificuldades Encontradas

Durante esse trabalho as maiores dificuldades foram por causadas pela falta de uma agenda do autor, pela demora no release da primeira versão do CMF, da falta de documentação e da grande dependência desse framework junto a BREW. Que acabaram em inviabilizar o *porting* de maneira a atender os dois fundamentos básicos portabilidade e facilidade de programação.

A uma seqüência de erro causado pelas ferramentas que levavam a constantes consultas a fóruns para solucionar problemas que não deveriam existir.

6.2. Trabalhos Futuros

Para dar continuidade a este trabalho, na verdade retomá-lo, será necessário, ou fica aqui sugerido, as seguintes modificações no *framework*:

- Remodelá-lo de forma a diminuir, ou de preferência abstrair completamente a dependência a uma plataforma
- Projetar o *framework* de forma a utilizar as características comuns às plataformas onde deseja implementar o mesmo.
- Utilizar-se de ferramentas de modelagem de programa orientados a objeto de tempo real (ex. *UML-RT* [39] ou *UML 2.0* [40])
- Melhorar a documentação tanto do código, como dentro dele.

Outro trabalho que pode ser realizado é o de estudar de forma extensa o processo de portar aplicativos entre plataformas, sejam entre PC, dispositivos móveis, ou mesmo Mainframes, para que casos como esse não venham a se repetir, ou pelo menos diminuam de forma considerável.

7. Referências

Capítulo 1

[01] Moore's Law, The Future - Technology & Research at Intel. Disponível em http://www.intel.com/technology/silicon/mooreslaw/. (30/08/2005)

[02]. Edge: Work-Group Computing Report. "Motorola Introduces World's First Java 2 Platform, Micro Edition —J2ME— -Enabled Wireless Communication and Computing Platform - Product Announcement". 12 de junho de 2000

[03]. Time-to-market. Disponivel em http://www.investorwords.com/4986/time_to_market.html (30/08/2005)

[04]. Qualcomm BREW Home. Disponivel em http://brew.qualcomm.com/brew/en/ (30/08/2005)

[05]. Microsoft Windows Mobile. Disponivel em http://www.microsoft.com/windowsmobile/default.mspx (30/08/2005)

Capítulo 2.

[06] Sun Microsystem. Disponivel em http://www.sun.com/ (30/08/2005)

[07] Motorola. Disponivel em http://www.motorola.com/ (30/08/2005)

[08] Datasheet Java 2 Platform, Micro Edition. On the Web sun.com/software.

[09] Java Technology. Disponivel em http://java.sun.com/ (30/08/2005)

[10] Java Community Process. http://jcp.org/en/home/index (30/08/2005)

[11] BREW and J2mE - A complete Wireless Solution for Operators Committed to Java. White Paper. 2003. QUALCOMM Internet Service. 2003.

[12] QUALCOOM Develepor Resources. Disponivel em http://brew.qualcomm.com/brew/en/developer/resources/dev_resources.html (30/08/2005)

[13] BREW® APIs: Version Released In. Qualcomm Internet Services. San Diego, CA. EUA. 2003

[14] Kernighan, B. W. e Ritchie, D. M. .The C Programming Language, Prentice-Hall: Englewood Cliffs, NJ, EUA 1978.

[15] Microsoft Corporation. Visual C and C++ 6.0. MSDN Library. Disponivel em http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvc60/html/msdn_vc6mgrtn.asp (30/08/2005)

[16] Product Information for Visual Studio .NET 2003. Microsoft Developer Network. Disponivel em http://msdn.microsoft.com/vstudio/productinfo/default.aspx (30/08/2005)
[17] TRUE BREW Process Overview. QUALCOOM Inc. San Diego, CA. EUA. 17 de Dezembro de 1998
[18] Symbian Ltd Overview. Disponivel em http://www.symbian.com/about/about.html (30/08/2005)
[19] Symbian Application Development. White Paper. Sonera MediaLab. 27 de janeiro de 2003.
[20] Series 60 Platform. Disponivel em http://www.nokia.com/nokia/0,8764,46827,00.html (30/08/2005)
[21] Symbian OS Version 7.0 product sheet . Disponivel em http://www.symbian.com/technology/symbos-v7x.html (30/08/2005)
[22] Microsoft Corporation. "Comparison of Windows CE .NET 4.2, Pocket PC 2002, and Windows Mobile 2003 Software for Pocket PCs". MSDN Library. Agosto de 2003.

Capítulo 3.

[23] Johnson, R. E., Foote, B., Designing Reusable Classes, Journal of Object-Oriented Programming, Vol. 1, No. 2, June 1988, pp. 23-35
[24] Mattson. M. Evolution and Composition of Object-Oriented Frameworks. Ph.d Thesis. University of Karlskrona/Ronneby. 2000.
[25] Gamma,E. , Helm. R., Johnson, R., Vlissides, J. (The Gang of Four). Design Patterns Elements of Reusable Object-Oriented Software. Addison-Wesley. Agosto de 1994.
[26] Stroustrup, B. ,The C++ Programming Language, Addison-Wesley: Reading, Mass., EUA. 1986.
[27] Barros. T. Developer Guide - CESAR Mobile Framework. C.E.S.A.R., Recife, PE, Brasil. 2005

Capítulo 4.

[28] Microsoft Corporation. What's New for Developers in Windows Mobile-based Pocket PCs. MSDN Library. Junho de 2003
[29] Microsoft Corporation. What's New for Developers in Windows Mobile 2003 Second Edition Software. MSDN Library. Março de 2004.

[30] Microsoft Corporation. What's New for Developers in Windows Mobile 5.0. MSDN Library. Maio de 2005

[31] .NET Compact Framework. Disponivel em http://msdn.microsoft.com/smartclient/understanding/netcf/ (30/08/2005)

[32] Microsoft Corporation. Introduction to Development Tools for Windows Mobile-based Devices.MSDN Library. Maio de 2005

[33] Visual Studio 2005 Developer Center. Disponivel em http://lab.msdn.microsoft.com/vs2005/ (30/08/2005)

[34] Microsoft Corporation. Embedded Visual C++ Programmer's Guide. MSDN Library. 2002

[35] ARM. Disponivel em http://www.arm.com/ (30/08/2005)

[36] Boling, D.Programming Microsoft Windows CE. NET 3rd Edition. Microsoft Press. 2003.

Capítulo 5

[37] Programming languages - C. ISO/IEC 9899:1990. ISO. 1990.

[38] Programming languages - C++. ISO/IEC 14882:2003. ISO. 2003.

Capítulo 6

[39] Schätz, B. . "UML-RT - die Lösung für eingebettete Software?". In: Proceedings of Modellierung 2004. Lecture Notes in Informatics. Primaveira, 2004

[40] Hause, M., Thom, F. Building Embedded Systems with UML 2.0/SysML. White Paper. Artisan Software. Paris 2005.