Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Informática

Desenvolvimento de Software para Dispositivos Móveis Baseados na Plataforma maemo™

Raul Fernandes Herbster

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Engenharia de Software
Linha de Pesquisa: Sistemas Embarcados

Angelo Perkusich
(Orientador)
Hyggo Oliveira de Almeida
(Orientador)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

H417d

2008 Herbster, Raul Fernandes

Desenvolvimento de software para dispositivos móveis baseados na plataforma maemo ™ / Raul Fernandes Herbster.— Campina Grande, 2008.

185 f..: il.

Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.

Referências.

Orientadores: Profº Dr. Angelo Perkusich, Profº Dr. Hyggo Oliveira de Almeida.

1. Engenharia de Software. 2. Dispositivos Móveis 3. Plataforma maemo I. Título.

CDU – 004.41 (043)

"DESENVOLVIMENTO DE SOFTWARE PARA DISPOSITIVOS MÓVEIS BASEADOS NA PLATAFORMA MAEMO(TM)"

RAUL FERNANDES HERBSTER

DISSERTAÇÃO APROVADA EM 23.07.2008

PROF. ANGELO PERKUSICH, D.Sc
Orientador

PROF. HYGGO OLIVEIRA DE ALMEIDA, D.Sc
Orientador

PROF. ANTONIO MARCUS NOGUEIRA DE LIMA, Dr.
Examinador

PROF. EVANDRO DE BARROS COSTA, D.Sc
Examinador

CAMPINA GRANDE – PB

Resumo

A crescente popularização de dispositivos móveis tem aumentado o interesse da indústria em produzir novos produtos que atendam aos requisitos dos usuários, cada vez mais exigentes. Aliado ao crescimento no consumo e à adoção de grandes empresas por produtos baseados em soluções *open source*, a Nokia lançou os produtos da linha Internet Tablet os quais utilizam a plataforma maemo, baseada no sistema operacional Linux. Documentação e ferramentas que facilitem o desenvolvimento são elementos fundamentais para a adesão de um grande número de desenvolvedores. Considerando tal fato, a comunidade maemo carece de i) um material didático com informações dispostas de maneira organizada e concisa para ajudar no aprendizado da plataforma maemo (arquitetura e desenvolvimento) e ii) ambientes de desenvolvimento com funcionalidades que facilitem a implementação, compilação, execução e testes (depuração e testes de unidade) de aplicações maemo. Neste trabalho, serão apresentados um documento que será utilizado para a publicação de um livro sobre a plataforma maemo, bem como duas ferramentas bastante importantes para o desenvolvimento de aplicações maemo: ESbox e PluThon. Além dessas contribuições, também é descrito um compilador JIT, desenvolvido também durante este trabalho, que utiliza o framework de compilação LLVM para processadores baseados na arquitetura ARM.

Abstract

The large adoption of mobile devices has been taken into account by the industry to produce new products that satisty requirements of the users. The users are becoming even more demanding. Considering the fact that the demanding of such devices is growing very fast and also the adoption of new open source based solutions by important mobile device companies, Nokia has released a new product line: the Internet Tablets. Such devices use the maemo platform, a Linux-based solution for embedded systems. Documentation and tools that make application development easier are essential elements for a massive adoption of the platform by the developers and, as a result, the success of a certain technology. Considering this fact, the maemo community lacks of i) a teaching material arranged in an organized and concise way to help newbies on maemo platform (concepts related to architecture and development) and ii) development environments that help the implementation, compilation, execution and tests (debugging and unit tests) of maemo applications. In this work, it is presented a document that is going to be used to publish a book on maemo platform, and also two very important tools to maemo application development: ESbox and PluThon. Besides these contributions, it is also described a JIT compiler, implemented during this work, for ARM-based processors that use LLVM compilation framework.

Agradecimentos

Agradeço a Deus pela força e coragem para ter chegado até este momento de minha vida.

À minha família, em especial meus adoráveis pais Pedro e Elvira, meus amados irmãos Adolfo e Yolanda, meus saudosos avós (Antônio, Zilda, Lauro e Iolanda), meus queridos tios e tias (em especial Albertina, por exercer o papel de minha mãe em Campina Grande) e todos os meus primos.

Aos meus orientadores Angelo Perkusich e Hyggo Almeida, pelas tarefas, problemas, soluções, debates, revisões e sobretudo pela orientação, compreensão e força prestados durante a execução deste trabalho.

A Marcos Morais, pelas conversas, idéias, dicas, ensinamentos, exemplos, soluções e por tudo mais que ele representa para os veteranos do Laboratório Embedded.

A todos os meus colegas e amigos do Laboratório Embedded, em especial da sala 106: Kyller, Mateus, Gutemberg, Saulo, Paulo Sausen, Walter, Taciana, Zé Luís, Danilo, Olympio, Mário, Ádrian, Diêgo, Yuri, André, Glauber, Leandro Sales, Leandro Leal e Marcus Fábio.

Quero também agradecer a amigos que me incentivaram bastante durante a realização deste trabalho e que, sem eles, o sucesso não seria completo: James, Makelli, Régis, Marizilda, Elias, André, Camila, Ana Esther, Ianna, Mirela, Janine e Amanda.

À Capes, pelo financiamento.

A todos os servidores e funcionários da DSC e COPIN pelo apoio.

Agradeço também a todos os meus ex-professores que serviram como bons ou maus exemplos de profissionais e cidadãos.

Conteúdo

Lista de Símbolos

ABI - *Application Binary Interface*

ALSA - *Advanced Linux Sound Architecture*

API - *Application Programming Interface*

ARM - *Advanced RISC Machine*

ARMEL - Nome utilizado para indicar a *EABI little endian ARM*

BT - *Bluetooth*

EABI - *"Embedded" ABI*

EAP - *Extensible Authentication Protocol*

ESD - *Enlightened Sound Daemon*

GPL - *GNU General Public License*

GTK+ - *GIMP Toolkit*

GUI - *Graphical User Interface*

I/O - *Input/Output*

INdT - Instituto Nokia de Tecnologia

JIT - *Just-In-Time*

LED - *Light Emitting Diode*

LGPL - *GNU Lesser General Public License*

miniSD - *mini Secure Digital Memory Card*

MIME - *Multipurpose Internet Mail Extension*

MMC - *Multimedia Card*

OMAP - *Open Multimedia Application Platform*

OSSO - *Open Source Software Operations*

PAI - Ponto de Acesso à Internet

PDS - *Personal Digital Assistant*

PDS - Processador Digital de Sinais

PPP - *Point-to-Point Protocol*

RS-MMC - *Reduced Size Multimedia Card*

RPC - *Remote Procedure Call*

SD - *Secure Digital Memory Card*

SSL - *Secure Sockets Layer*

TLS - *Transport Layer Security*

TI - *Texas Instrument*

WLAN - *Wireless Local Area Network*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Sistemas embarcados são sistemas microprocessados no qual o computador é completamente encapsulado ou dedicado ao dispositivo ou sistema que ele controla [90]. Ao contrário de um computador de propósito geral (por exemplo, um computador pessoal), um sistema embarcado realiza uma ou poucas tarefas pré-definidas, geralmente com requisitos bastante específicos. Uma vez que o sistema é dedicado a tarefas específicas, os projetistas podem otimizar as aplicações, reduzindo o custo e o tamanho do produto. Sistemas embarcados são produzidos em massa, tomando proveito de economias em escala.

Personal Digital Assistant ou *Handheld computers* (PDAs), por exemplo, também são considerados sistemas embarcados, devido à natureza do projeto de hardware, mesmo que possuam mais recursos, permitindo aplicações mais complexas [90]. Fisicamente, sistemas embarcados podem variar de dispositivos portáteis como MP3 *players* até sistemas distribuídos complexos, tais como controladores em uma planta industrial de chão de fábrica.

O processo moderno de desenvolvimento de software engloba diversas tarefas realizadas iterativamente, tais como projeto arquitetural, documentação, codificação, teste, integração, etc. A realização destas tarefas com o uso de ferramentas independentes pode consumir muito tempo e esforço, principalmente ao se considerar a mudança constante na implementação e no modo de utilização de tais ferramentas.

Em um contexto mais específico de desenvolvimento, aplicações para sistemas embarcados possuem diversas características peculiares que as diferem de outros tipos de aplicação. Por exemplo, o desenvolvimento de um produto é tipicamente realizado em um ambiente de propósito geral, com sua própria arquitetura de hardware e sistema operacional. Porém,

o produto final é voltado para a execução em uma plataforma diferente. O ambiente alvo da aplicação é geralmente mais complexo em comparação à plataforma utilizada para o desenvolvimento, pois possui uma série de restrições que devem ser respeitadas, por exemplo, memória limitada se comparado com o ambiente de desenvolvimento, interfaces de usuário restritas, gerência de gasto de energia, elementos de mobilidade, etc [91].

Para gerenciar os diversos módulos de hardware do dispositivo móvel, é necessário um sistema operacional com o intuito de prover abstrações para gerenciamento e acesso a recursos tanto para o desenvolvedor quanto para o usuário. Atualmente, existem alguns sistemas operacionais especialmente desenvolvidos para dispositivos móveis, por exemplo: Symbian [21; 38; 83], Linux embarcado e Windows CE [67; 16]. Geralmente, cada um desses sistemas é utilizado em uma determinada linha de produtos de uma fabricante de dispositivos móveis. Por exemplo, os celulares Nokia utilizam o sistema operacional Symbian em seus celulares (S60 e *smart phones* [69], por exemplo), enquanto que, na linha de produtos *Internet Tablet* [28], utiliza-se a plataforma maemo (uma distribuição Linux embarcada).

Maemo [58] é uma plataforma *open source* desenvolvida pela Nokia e baseada no sistema operacional GNU/Linux. Oficialmente lançada em março de 2005, a plataforma oferece recursos para o desenvolvimento de soluções atrativas para facilitar a conectividade do usuário final com diversos serviços disponibilizados em uma rede ou na Internet (*media centers*, sistemas de comunicação, VoIP, etc.). O maemo foi especialmente desenvolvido para a linha de produtos da Nokia denominada *Internet Tablets*, dispositivos móveis com uma série de recursos (por exemplo, interfaces de comunicação Wi-Fi [76] e Bluetooth [17]). Dessa forma, é possível desenvolver aplicações embarcadas atrativas em termos de funcionalidade e interface gráfica para o usuário.

O número de desenvolvedores de aplicações maemo cresceu consideravalmente, devido também à evolução da plataforma que agregou diversas funcionalidades importantes (suporte a diversas APIs de comunicação, interface gráfica e multimídia) e ao nível de estabilidade alcançado. Contudo, para o crescimento maior da comunidade de desenvolvedores, é essencial a existência de documentação e ferramentas que auxiliem o desenvolvimento de aplicações para a plataforma, facilitando o aprendizado de desenvolvedores que desconhecem as tecnologias utilizadas. Atualmente na versão 4.1 (*codinome* Diablo), a plataforma maemo possui um site oficial [58] no qual é possível encontrar vários recursos de desenvolvimento, tais

como tutoriais, repositórios, SDKs e notícias. Contudo, não há uma fonte na qual as informações sobre desenvolvimento de aplicações maemo estejam organizadas em uma seqüência que facilite o entendimento por parte do programador.

1.1 Problemática

Como dito anteriormente, documentação e ferramentas que facilitem o desenvolvimento são elementos fundamentais para a adesão de um grande número de desenvolvedores e, conseqüentemente, o sucessso de uma determinada tecnologia [11]. Para que desenvolvedores de outras tecnologias passem a contribuir ou até adotar a plataforma maemo como base de suas soluções, é importante um suporte de qualidade, disponível através de documentação e também ferramentas que auxiliem o desenvolvimento. Os documentos oficiais estão disponíveis no site da plataforma [58]: tutoriais, *how-to*s e outras informações. Entretanto, não há nenhum material que apresente de forma organizada e sistematizada as informações já existentes com o intuito de facilitar o estudo e o entendimento por parte do programador. Além disso, algumas informações importantes não estão presentes em [58], tampouco em outra fonta, tornando mais difícil a aprendizagem do programador.

Inseridas nesse contexto, também estão os ambientes de programação que facilitem o desenvolvimento de aplicações e possam ser utilizados, também, como ferramenta para auxílio no aprendizado de uma determinada tecnologia. A plataforma maemo não dispõe de ambientes integrados de desenvolvimento (*IDE - Integrated Development Environment*) para facilitar a implementação de aplicativos. O programador tem, como recursos, apenas editores de texto simples (vi, vim, gedit) e ferramentas baseadas em linha de comando utilizadas para, por exemplo, documentação, compilação, link-edição, implantação, depuração e controle de versão. Com isso, além de dificultar o aprendizado da plataforma maemo, também deixa o processo de desenvolvimento menos produtivo e suscetível a um maior número de falhas.

Dessa maneira, aliando-se um material suficientemente claro e organizado de forma a facilitar a compreensão da plataforma maemo (desenvolvimento de aplicações e arquitetura) com um ambiente de desenvolvimento gráfico que auxilie nas várias tarefas envolvidas no processo (documentação, codificação, compilação, depuração e implantação), é possível transferir conhecimentos relativos à plataforma maemo de forma mais fácil e sistematizada.

1.2 Objetivo

Este trabalho tem como objetivo principal produzir um livro com informações, dispostas de maneira organizada, sistematizada e concisa, bastante úteis para o aprendizado da plataforma maemo (desenvolvimento e arquitetura). O que se procura não é apenas produzir mais um documento, mas sim, um material que reúna conhecimento oriundo de várias fontes (artigos técnicos, artigos científicos, documentação existente em [58] e em outros *sites*, bem como experiências obtidas que tornam explícito em conhecimento tácito o desenvolvimento de aplicativos maemo) e o disponibilize de maneira a facilitar o claro entendimento sobre a plataforma. Além disso, neste trabalho apresenta-se o projeto e implementação de IDEs que auxiliem na execução das diversas fases envolvidas no processo de desenvolvimeno de aplicações maemo. Dessa forma, além de facilitar a aprendizagem das atividades relacionadas à programação de aplicativos maemo, as ferramentas também trazem ganhos em qualidade, tempo e esforço durante o processo de desenvolvimento.

Dessa maneira, são apresentadas duas contribuições centrais para abordar a problemática descrita na Seção 1.1, quais sejam:

1. Definição, organização e apresentação do conteúdo de um livro que contemple a arquitetura (módulos de gerência de recursos, carregamento de aplicações, subsistemas de conectividade, comunicação, interface gráfica e multimídia) e o processo de desenvolvimento de software da plataforma maemo (fases do processo de desenvolvimento e ferramentas envolvidas). Além disso, outros aspectos não-funcionais também são discutidos, como certificação de qualidade e segurança das aplicações;

2. Contribuições para o desenvolvimento de ambientes, ESbox [22; 40] e PluThon [23], que facilitam a execução do processo de desenvolvimento de aplicações maemo, pois todas as fases podem ser realizadas em um único ambiente, sem a necessidade da troca constante de interfaces gráficas. Ambas as ferramentas foram adotadas como ambientes de desenvolvimento oficial da plataforma maemo e compõem o projeto *IDE Integration* [56].

1.3 Relevância

Um dos principais problemas recorrentes em grande parte das comunidades *free software* ou *open source* ainda é a carência de documentação. Na plataforma maemo, é possível encontrar diversas informações relativas ao desenvolvimento de aplicações. Contudo, elas estão disponibilizadas de maneira não sistematizada, sem a devida estrutura e organização que facilitem o aprendizado. Não há nenhum livro existente sobre a plataforma maemo que procure minimizar esses problemas, tampouco um material de outra natureza (tutorial *on-line*, por exemplo). Além disso, a carência de ferramentas que facilitem o processo de desenvolvimento e ajudem o processo de aprendizagem da plataforma maemo, a partir de um ambiente gráfico com várias ações facilmente acessíveis, agrava mais a situação. Com a disponibilização do livro e também dos ambientes integrados de desenvolvimento, o programador passa a ter ferramentas suficientes para aprender sobre a arquitetura e também como desenvolver aplicações para a plataforma maemo. Os resultados propostos também são uma grande contribuição para a comunidade maemo, pois incrementam o número de recursos disponíveis para o desenvolvedor, aumentando ainda mais o número de adeptos da plataforma.

1.4 Organização

Este trabalho possui dois tipos de capítulos: aqueles que são parte única e exclusivamente de uma dissertação de mestrado (Capítulos 13 e 14) e os demais que fazem parte do livro (Capítulos 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 e 12). Este trabalho está organizado da seguinte maneira:

Capítulo 2 Descreve-se um resumo da plataforma maemo: histórico, arquitetura, principais módulos e bibliotecas do sistema;

Capítulo 3 Neste capítulo, o mecanismo de execução de aplicações maemo, bem como o ciclo de vida das aplicações, são descritos em detalhes;

Capítulo 4 Ambientes de desenvolvimento integrados que auxiliam no desenvolvimento de aplicações maemo (ESbox [22] e PluThon [23]) são descritos no Capítulo 12;

Capítulo 5 O principal mecanismo de comunicação entre aplicações da plataforma maemo, o sistema D-Bus [31], é apresentado neste capítulo;

Capítulo 6 A plataforma maemo oferece alguns frameworks para o desenvolvimento de aplicações gráficas, por exemplo, GTK+ [72; 47] e Hildon. Neste capítulo, os frameworks considerados padrão da plataforma (GTK+/Hildon) são apresentados, bem como outras soluções disponíveis (Evas/Edje [26] e SDL [43]);

Capítulo 7 O principal componente que possibilita o desenvolvimento de aplicações multimídia é o framework GStreamer [37]. Neste capítulo, são descritos outros módulos do subsistema de multimídia da plataforma maemo, bem como o framework GStreamer;

Capítulo 8 Conectividade é uma funcionalidade importante na plataforma maemo, pois uma das principais metas dos Internet Tablets é oferecer conexão à Internet sem limitações. Neste capítulo, os principais módulos do subsistema de conectividade da plataforma são descritos, bem como algumas bibliotecas de uso comum;

Capítulo 9 Segurança é um aspecto importante em aplicações para dispositivos móveis, uma vez que grande parte dos dados trafegam em redes desconhecidas. Assim, é importante que o desenvolvimento de aplicações maemo considerem requisitos de segurança, os quais são discutidos neste capítulo.

Capítulo 10 Embora não exista uma entidade certificadora de qualidade para aplicações maemo, é possível obter um grau considerável de qualidade seguindo atividades discutidas neste capítulo;

Capítulo 11 O desenvolvimento de aplicações maemo, assim como outras tecnologias, também necessita de algumas ferramentas, por exemplo, GCC [4; 36] e GNU Autotools [45]. Neste capítulo, são descritas as ferramentas comumente utilizadas, bem como os problemas existentes no desenvolvimento de aplicações maemo;

Capítulo 12 ESbox e PluThon, duas ferramentas desenvolvidas no Laboratório Embedded, na Universidade Federal de Campina Grande, e que também foram adotadas pelo projeto maemo como ambientes de desenvolvimento da plataforma maemo, são apresentadas neste capítulo;

Capítulo 13 Durante este trabalho, também foram implementadas outras contribuições para projetos *open source* e voltados para o desenvolvimento de aplicações para dispositivos móveis. Neste capítulo, é apresentado um compilador JIT (*Just-In-Time*), utilizando-se o framework de compilação LLVM [49; 50; 2], para processadores baseados em ARM [6; 79; 82];

Capítulo 14 São apresentadas as conclusões do trabalho, descrevendo suas contribuições e trabalhos futuros.

Capítulo 2

Plataforma maemo

Neste capítulo, será apresentada a plataforma maemo, uma das primeiras tecnologias baseadas em Linux embarcado utilizada em um produto comercial, o Internet Tablet, que alcançou uma considerável aceitação tanto dos usuários finais quanto dos desenvolvedores. Maemo se tornou bastante popular e cresceu juntamente com uma comunidade ativa e produtiva. A plataforma é um exemplo de como Linux pode ser utilizado em produtos para usuários finais, resultando em uma interface clara e intuitiva, promovendo funcionalidades que tornam a plataforma ainda mais atraente.

2.1 Histórico

Anunciada no final de 2004 e lançada em 2005, maemo é uma plataforma aberta para desenvolvimento de aplicações e uma inovação tecnológica para dispositivos portáteis. Foi criada pela Nokia como parte do processo de desenvolvimento dos Internet Tablets e, logo após, teve o código aberto e liberado para a comunidade de desenvolvedores. A plataforma maemo trás uma maneira fácil e otimizada de desenvolver aplicações para dispositivos móveis.

A palavra foi criada usando o gerador de senhas (`pwgen`), sendo escolhida porque soava bem e aparentemente não significava nada ofensivo em nenhuma linguagem existente.

Em 25 de maio de 2005, a Nokia lançou a primeira versão do SDK de desenvolvimento de aplicações para a plataforma. Logo após, a versão 2.0 do SDK maemo foi lançado, com duas revisões posteriores (codinomes Mistral, Scirocco e Gregalle, respectivamente). No início de 2007, o SDK maemo 3.0 (codinome Bora) foi lançado juntamente com o Internet

Tablet Nokia N800. Finalmente, em novembro de 2007, a versão 4.0 da plataforma maemo (codinome Chinook) foi anunciada e lançada juntamente com o novo Internet Tablet Nokia N810. Atualmente, o SDK maemo se encontra na versão 5.0 (codinome Diablo). Em março de 2008, a Nokia lançou uma nova versão limitada do IT N810 com suporte à interface WiMax. A diferença entre as versões do SDK reside basicamente nas bibliotecas e ferramentas, as quais podem ser removidas, adicionadas ou atualizadas. A linha de evolução da plataforma maemo [85] está ilustrada na Figura 2.1.

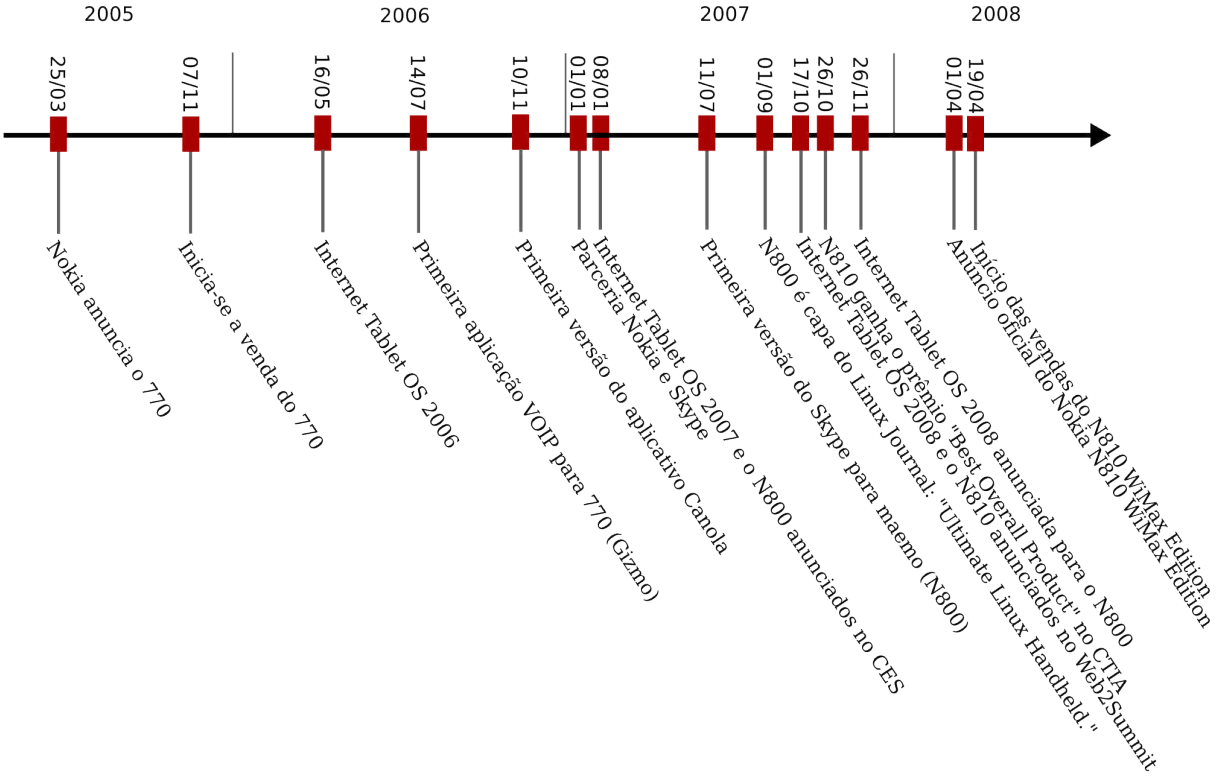

Figura 2.1: Linha de Evolução da Plataforma maemo.

2.2 Visão Geral

A plataforma maemo é o resultado do esforço da Nokia em desenvolver um dispositivo móvel baseado em Linux e tecnologias *open source*. A motivação por trás do maemo e de sua disponibilização à comunidade está no estímulo ao desenvolvimento e adoção da tecnologia Linux. Maemo provê um rico ambiente de desenvolvimento, construção e testes. O ambiente de desenvolvimento (*HOST*) executa os mesmos softwares que estão dispobilizados

no ambiente alvo (*TARGET*), eliminando a necessidade de emulação do hardware alvo no ambiente hospedeiro, provendo, assim, um ambiente de testes mais fiel. É importante a simplificação do ambiente de desenvolvimento para o dispositivo móvel, uma vez que maximiza o aprendizado e a produtividade. As principais funcionalidades de plataforma maemo são:

- Permite o desenvolvimento de aplicações usando o framework de aplicações Hildon;
- Provê um ambiente de desenvolvimento para aplicações que necessitem de conexão à rede;
- Provê um ambiente de testes na arquitetura x86;
- Permite a depuração na arquitetura x86;
- Possibilita a configuração de diversos ambientes de desenvolvimento;
- Atulização progressiva dos ambientes de desenvolvimento;
- Extensa documentação para os desenvolvedores através do site `http://www.maemo.org`;
- Listas para discussão, anúncios e suporte.

2.3 Sistema Operacional Linux

Linux é um sistema operacional *multi-threading*, multi-usuário e gratuito. Portado para as mais diversas arquiteturas (x86, x86-64, MIPS, ARM, PowerPC, Alpha, Sparc, por exemplo), o sistema é cada vez mais utilizado em ambientes embarcados [92]. Até mesmo produtos comerciais de grandes empresas já utilizam Linux como sistema operacional: Motorola $RAZR^2$ V8, A780, E680 e A1200; Openmoko Neo1973; Nokia 770, N800 e N810.

2.3.1 Núcleo

O núcleo é a parte mais importante do sistema Linux. Ele controla recursos, memória, processos e seu acesso à CPU bem como é responsável pela comunicação entre os componentes de hardware e software. O núcleo provê uma camada de acesso a recursos do sistema tais

como memória, CPU e dispositivos de E/S. A comunicação com o núcleo é realizada através de *chamadas de sistema*, funções genéricas utilitárias que realizam o trabalho com diferentes dispositivos, gerência de processos e de memória. A vantagem na utilização de chamadas de sistema está na sobrecarga de suas funções. Por exemplo, não importa o dispositivo de E/S utilizado, pois a função permanece a mesma [81].

O núcleo divide a memória virtual em dois espaços: o espaco do usuário e o espaço do núcleo. O espaço do usuário é a área de memória onde estão as aplicações que executam em modo usuário. Neste caso, a aplicação pode acessar dispositivos de hardware, memória virtual ou outro serviço do núcleo apenas através de chamadas de sistema. O espaço do núcleo é reservado para controladoras de dispositivos, extensões do núcleo e para o próprio núcleo. A Figura 2.2 ilustra os espaços de memória virtual existentes.

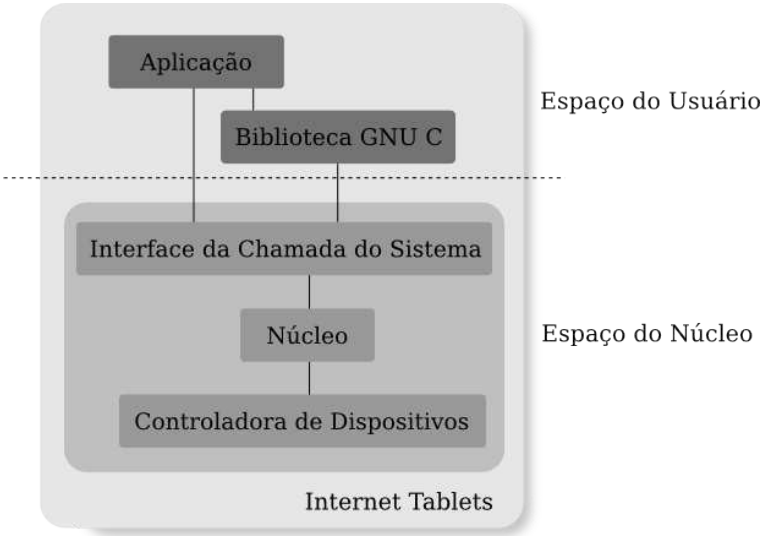

Figura 2.2: Espaços de memória.

O núcleo provê mecanismos para carregamento dinâmico de módulos. Os módulos permitem a extensão das funcionalidades do núcleo - tais como um novo protocolo de comunicação ou uma nova controladora de dispositivo - sem recompilar ou reiniciar todo o núcleo.

2.3.2 Processos

Um processo é um programa em execução, constituído basicamente de uma área de texto (na qual se localiza o código a ser executado), uma área de dados (onde são armazenadas as

variáveis) e uma área de memória (heap e pilha) que podem se expandir caso necessário [81; 84]. A estrutura tradicional de um processo é ilustrada na Figura 2.3. Tal estrutura de dados pode variar de um sistema operacional para outro. Um processo no sistema Linux possui uma série de características, tais como:

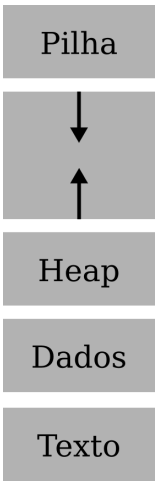

Figura 2.3: Estrutura padrão de processos.

PID O identificador numérico do processo;

PPID O identificador do processo pai;

Estado O estado atual do processo;

Arquivos abertos A lista com os arquivos abertos pelo processo;

TTY O terminal com o qual o processo está conectado.

A criação de processos é realizada pelo núcleo, portanto, através de chamadas de sistema. As operações utilizadas para tal tarefa são `fork` e `exec`. Um processo pode criar um clone de si próprio através da operação `fork`. O processo criador do clone passa a ser pai do processo clonado e um novo PID é associado ao processo clonado. Os processos clonados dividem com o pai alguns dos elementos, tais como a área de memória. Portanto, para obter a estrutura de dados própria, deve-se invocar a chamada de sistema `exec` e assim obtem-se um processo novo com estrutura de dados própria. Os mecanismos de criação de um processo no sistema Linux é ilustrado na Figura 2.4.

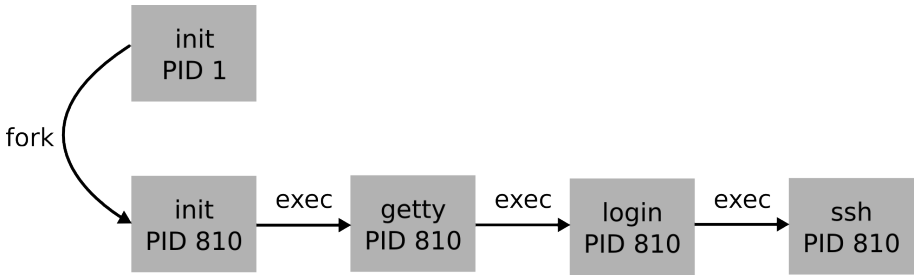

Figura 2.4: Mecanismo de criação de processos.

Tabela 2.1: Sinais comumente enviados aos processos no sistema Linux

Sinal	Valor numérico	Descrição
SIGTERM	15	Finaliza o processo normalmente
SIGINT	2	Interrompe o processo. Pode ser ignorado pelo processo
SIGKILL	9	Interrompe o processo. Não pode ser ignorado pelo processo
SIGUP	1	Usado por processos daemon, informam ao daemon para atualizar a configuração

Sempre que um processo é finalizado, ele envia um sinal numérico de retorno ao processo pai. O sinal de retorno é definido pelo programa, portanto, não há nenhum padrão. Geralmente, o código de retorno 0 é utilizado para determinar que um processo foi finalizado com sucesso. Os processos também podem ser finalizados através de envio de sinais a eles. No sistema Linux, existem mais de 60 sinais diferentes. Os mais utilizados estão listados na Tabela 2.1. Somente o usuário dono do processo (ou superusuário) pode enviar sinais ao processo.

2.3.3 Sistema de Arquivos

O sistema Linux segue o padrão de sistemas Unix, chamado *unified hierarchical namespace*, para organização de um sistema de arquivos. Todos os dispositivos e partições do sistema de arquivos parecem pertencer a uma única hierarquia. Na organização do sistema de arquivos, todos os recursos podem ser referenciados a partir do diretório raíz (identificado pelo caracter

/), e todo arquivo ou dispositivo existente no sistema também pode ser acessado a partir desse diretório.

É possível acessar diferentes recursos de sistema de arquivos dentro de um mesmo espaço de nomes. É necessário somente dizer ao sistema operacional a localização no espaço de nomes do sistema de arquivos onde é desejável que o recurso apareça. Essa tarefa é chamada de montagem (*mounting*), enquanto que a localização de espaço de nomes na qual se deseja inserir o sistema de arquivo ou recurso é também conhecido como ponto de montagem (*mounting point*).

O mecanismo de montagem permite o estabelecimento de espaço de nomes de maneira coerente, onde diferentes recursos podem ser sobrepostos de maneira normal e transparente. Por exemplo, supõe-se a necessidade de montar um cartão de memória que possui três diretórios distintos: `pastaa`, `pastab` e `pastac`. O conteúdo do cartão de memória deve estar disponível no diretório `/media/mmc1`. Além disso, o arquivo de dispositivo do cartão de memória está em `/dev/mmcblk0p1`. Deve-se executar o seguinte comando para indicar onde no espaço de nomes do sistema de arquivos se deseja montar o cartão de memória.

```
/ $ sudo mount /dev/mmcblk0p1 /media/mmc2
/ $ ls -l /media/mmc2
total 0
drwxr-xr-x 2 user group 1 2008-04-19 11:43 pastaa
drwxr-xr-x 2 user group 1 2008-04-19 11:43 pastab
drwxr-xr-x 2 user group 1 2008-04-19 11:43 pastac
/ $
```

Além dos dispositivos físicos, o sistema Linux também suporta sistemas de arquivos virtuais. Eles se comportam como sistemas de arquivo normais mas não representam dados persistentes, e sim, provêem acesso às informações do sistema, configuração e dispositivos. Dessa forma, o sistema operacional pode oferecer maior número de recursos e serviços como parte do espaço de nomes do sistema de arquivo. Por exemplo, o ***procfs*** é um sistema de arquivos virtual usado para acessar informações de processos do núcleo e é montado, geralmente, no diretório `/proc`.

Grande parte das distribuições Linux são baseadas no *Filesystem Hierarchy Standard* (FHS)[1], que é um padrão bastante detalhado e possui uma série de requisitos e guias para armazenamento de pastas e arquivos em sistemas operacionais Unix. Por exemplo, o padrão determina que o diretório `/sbin` contenha binários essenciais ao sistema, enquanto o diretório `/dev` deve possuir os arquivos de dispositivos.

Quando um sistema de arquivos é criado em uma partição, uma estrutura de dados que contém informações sobre os arquivos presentes é criada. Chamadas de *inodes*, essas estruturas armazenam um conjunto de dados importantes sobre o arquivo, por exemplo: o dono do arquivo, o grupo ao qual pertence o arquivo, as permissões relacionadas ao arquivo, o tamanho, datas de última leitura, de ultima modifição e de criação do arquivo.

As informações sobre o *inode* de um determinado arquivo podem ser conferidas utilizando o comando `stat`, conforme abaixo. Para realizar quaisquer operações sobre o arquivo, o núcleo consulta o *inode* referente.

```
/ $ stat /etc/group
File: `/etc/group'
Size: 851             Blocks: 8          IO Block: 4096   regular file
Device: 803h/2051d    Inode: 7979341     Links: 1
Access: (0644/-rw-r--r--)  Uid: (0/root) Gid: (0/root)
Access: 2008-04-21 12:48:47.000000000 -0300
Modify: 2007-12-04 20:47:18.000000000 -0300
Change: 2007-12-04 20:47:18.000000000 -0300
```

Múltiplos arquivos podem estar associados a um mesmo *inode*, criando os chamados *hard links*. O sistema de arquivo Linux também suporta *soft links*, ou *symbolic links*. Um *soft link* contém o caminho para o arquivo alvo no lugar de uma localização física no disco [81; 84].

O modelo de segurança do sistema de arquivo Linux é baseado no mesmo modelo Unix. Todo usuário do sistema possui um identificador (ID) e todo usuário pertence a um ou mais

[1]Filesystem Hierarchy Standard. Maiores detalhes disponíveis em `http://www.pathname.com/fhs/`

grupos, identificados por valores numéricos (ID do grupo). Todo arquivo pertence a um usuário e a um grupo de usuários. Há ainda uma terceira categoria: outros, os quais são os usuários que não são donos do arquivo e nem pertencem ao grupo dono do arquivo. A cada uma das três categorias, estão associadas três permissões que podem ser adquiridas ou removidas: escrita, leitura e execução. Para se verificar as permissões associadas ao arquivo, basta utilizar o comando abaixo:

```
/ $ ls -la /home/
drwxr-xr-x  3 root root 4096 2007-11-29 15:32 .
drwxr-xr-x 24 root root 4096 2008-04-10 19:37 ..
drwxr-xr-x 90 user user 4096 2008-04-21 21:34 user
```

2.4 Projeto da Plataforma

A plataforma é formada por bibliotecas presentes no ambiente *desktop* e também por elementos desenvolvidos especificamente para ambientes embarcados. Na Figura 2.5 estão presentes os principais elementos que compõem o sistema. A plataforma é baseada no sistema operacional GNU/Linux e no *desktop* GNOME. Maemo oferece ao desenvolvedores um ambiente de desenvolvimento de fácil utilização: o *framework* de aplicações Hildon. Ele é otimizado e especialmente desenvolvido para dispositivos com interface gráfica limitada (tamanho da tela pequena) e métodos de interação diferenciados (tela sensível ao toque). Além disso, ela traz contribuições importantes para a plataforma (por exemplo, gerenciamento de memória e carregamento de aplicações) e facilita o desenvolvimento de aplicativos.

É possível utilizar a plataforma também para portar aplicações já existentes no ambiente desktop. Maemo possibilita a criação de outros modelos de negócio no topo da plataforma, como por exemplo, a distribuição de aplicativos proprietários.

Atualmente, maemo executa sobre uma versão 2.6 do núcleo Linux. O espaço do usuário é ligado ao sistema utilizando a biblioteca GNU C (glibc). A plataforma maemo procura ser o mais compatível possível com sistemas Linux atuais, com o intuito de diminuir o tempo e esforço necessários para o porte de aplicações existentes e, também, o desenvolvimento de

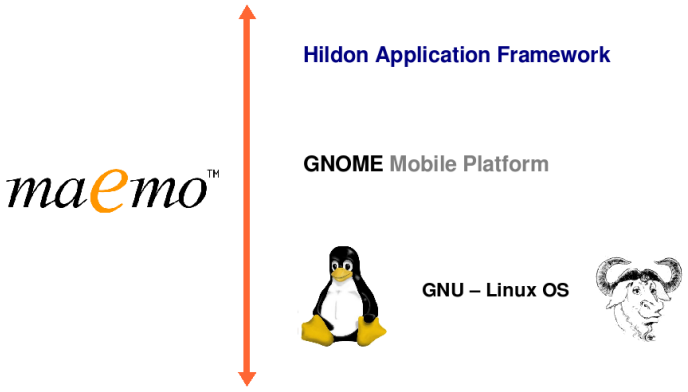

Figura 2.5: Projeto da Plataforma maemo.

novas aplicações para a plataforma maemo.

Com o intuito de facilitar a instalação, configuração, atualização e remoção de aplicativos, a plataforma provê um gerenciador de pacotes baseado na distribuição Debian (comandos dpkg).

2.4.1 Bibliotecas de Desenvolvimento

Conforme discutido no Capítulo 1, a plataforma maemo é baseada em uma distribuição Debian. Portanto, o desenvolvimento de aplicações maemo é bem semelhante ao desenvolvimento de aplicações Debian/Linux. Embora existam várias bibliotecas de desenvolvimento para serem utilizadas, algumas foram adotadas como padrão pela plataforma maemo. As principais bibliotecas de desenvolvimento são:

Biblioteca GNU C - glibc [74] É uma biblioteca C desenvolvida pelo projeto GNU. Provê funcionaliadades requeridas pelos padrões ASCII C, POSIX (1c, 1j e 1d) e também algumas descritas no padrão ISO C99. Todas aplicações que executam na plataforma maemo utilizam, direta ou indiretamente, esta biblioteca;

GLib Biblioteca bastante útil que provê vários tipos de dados portáveis: estrutura de dados, *macros, strings, framework* de desenvolvimento orientado a objetos, dentre outros. A biblioteca GLib bastante utilizada para tornar a aplicação independente de tipos de uma determinada plataforma;

GObject [47] Possibilita o desenvolvimento de aplicações C orientadas a objeto através de uma maneira flexível e eficiente;

GConf [32] Provê funcionalidades de gerenciamento de configurações, permitindo que aplicações salvem e recuperem o seu estado de maneira segura, sem a necessidade de arquivos de configuração;

Gnome-VFS [33] Abstrai a utilização do sistema de arquivos, possibilitando que a aplicação realize operações (salvar, ler, apagar e criar arquivos), sem a necessidade de conhecer a semântica entre os diversos dispositivos e serviços, ou seja, o desenvolvedor não precisa se preocupar se um arquivo está sendo lido de um servidor Web ou se um cartão de memória;

LibOSSO [57] Biblioteca que possui um conjunto de funções úteis e necessárias para aplicações maemo, por exemplo: mecanismos que permitem a conexão com o sistema D-Bus de maneira clara e eficiente; um sistema que permite a serialização do estado das aplicações, facilitando a mudança de contexto;

D-Bus É uma implementação de um serviço RPC (*Remote Procedure Call*), permitindo que processos troquem mensagens entre si. No sistema, há um servidor D-Bus responsável pela gerência da comunicação entre os vários processos. O servidor também é responsável pela notificação de eventos do sistema principal (estado da bateria, estado das interfaces de comunicação, etc.) para as aplicações.

Essas são as bibliotecas de integração da aplicação com a plataforma. Existem outras bibliotecas as quais serão citadas nas seções posteriores e discutidas detalhadamente nos próximos capítulos.

2.4.2 Interface Gráfica

O módulo de interface gráfica é baseado no *framework* `GNOME`, principalmente a biblioteca `GTK+`. Com o intuito de suprir as necessidades da plataforma, desenvolveu-se a biblioteca Hildon, baseada em GTK+. Tal biblioteca possui um conjunto de elementos gráficos úteis (seletores de arquivos, janelas, dentre outros) e mudanças (cores claras, fontes maiores, cantos arredondados) no *layout* que tornam as aplicações mais atraentes nos *Internet Tablets*.

As principais bibliotecas utilizadas no desenvolvimento de aplicações gráficas na ferramenta são:

GTK+ Biblioteca multi-plataforma para criação de interfaces gráficas. Os elementos gráficos GTK+ são chamados de *widgets*. A biblioteca também implementa temas de interface (determina um conjunto de modelos de comportamento e características dos gráficos). GTK+ é baseado em outras bibliotecas: GLib, GDK, Pango e ATK;

- **GDK [47]** É uma biblioteca de gráficos que atua como *wrapper* de funções de manipulação de janelas e gráficos baixo nível;
- **Pango [47]** Biblioteca portável utilizada para visualização de textos nas mais diferentes codificações;
- **ATK [47]** Provê métodos genéricos para implementação de funcionalidades de suporte a pessoas com necessidades especiais;

Cairo [47] Gera gráficos 2D de maneira consistente, podendo utilizar aceleração de hardware quando disponível;

Hildon Biblioteca que provê um conjunto de elementos de interface gráfica e temas especialmente criados para a plataforma maemo, com melhorias sobre o GTK+ (por exemplo, fontes maiores e cores mais claras para facilitar a visualização por parte do usuário). É utilizada na criação de UI para a plataforma.

Embora a API de desenvolvimento de inteface gráfica padrão seja GTK/Hildon, há outras possibilidades. A plataforma também possui suporte para o desenvolvimento de aplicações baseadas em SDL (*Simple Direct Layer*) ou em Evas/Edje. SDL é uma biblioteca desenvolvida pela Loki Entertainment, com o foco no desenvolvimento de jogos para Linux. Possui diversas funcionalidades que facilitam a criação de jogos e de gráficos de baixo nível. Evas e Edje são duas bibliotecas mantidas pela EFL (*Enlightenment Foundation Library*) [26] e são utilizadas para implementar aplicações que necessitam de interfaces gráficas mais elaboradas, agregando valor ao produto final.

Detalhes relativos aos frameworks de interface gráfica da plataforma, são discutidos no Capítulo 6.

2.4.3 Áudio e Vídeo

O subsistema de multimídia da plataforma maemo é baseado no framework GStreamer, Helix e EsoundD, que foram modificados para agregar a solução baseada na utilização de processador digital de sinais (PDS) da plataforma OMAP [44; 53], presentes nos Internet Tablets. Os detalhes sobre o subsistema de multimídia são discutidos no Capítulo 7. Abaixo estão listados os principais frameworks utilizados em aplicações multimídia da plataforma maemo.

ALSA [43] Advanced Linux Sound Architecture provê funcionalidades de áudio. O framework também tem compatibilidade com outras bibliotecas mais antigas, como o Open Sound System (OSS);

ESD [43] Enlightened Sound Daemon é um servidor de áudio para o Linux;

GStreamer É um framework para o desenvolvimento de aplicações multimídia. Foi desenvolvido com o intuito de prover uma solução multimídia de qualidade para a plataforma Linux;

Video4Linux [86] É uma API para captura de vídeo, portado para a plataforma maemo e integrado com a câmera de alguns Internet Tablets. Video4Linux é bem integrado com o núcleo do Linux.

2.4.4 Conectividade

A plataforma maemo oferece suporte a três interfaces de comunicação: USB, Wi-Fi (IEEE 802.11) e Bluetooth. Mais recentemente, foi lançada uma versão limitada do IT N810, que oferece também a interface WiMax (IEEE 802.16).

Sockets e TCP/IP [19; 13] Bibliotecas padrão disponibilizadas pelo Linux;

BlueZ [15] Implementação das especifições do padrão Bluetooth™para Linux;

OpenSSL [87] Biblioteca que provê uma camanda de segurança de rede para criptografia de dados;

cURL [1] Utilizada para implementação de clientes para transferência via URL: HTTP, HTTPS, FTP, FTPS, etc.

Detalhes relativos aos frameworks de conectividade, são discutidos no Capítulo 8.

2.4.5 Outros Componentes

A plataforma possui um número considerável de outras bibliotecas utilizadas no desenvolvimento de aplicações, conforme ilustrado na Figura 2.6. Contudo, tais módulos não são comumente utilizados e não serão tratadas nesse trabalho por se tratarem de elementos bem específicos. Para maiores detalhes, informações e documentação sobre os componentes extras estão disponíveis no site oficial da plataforma [58].

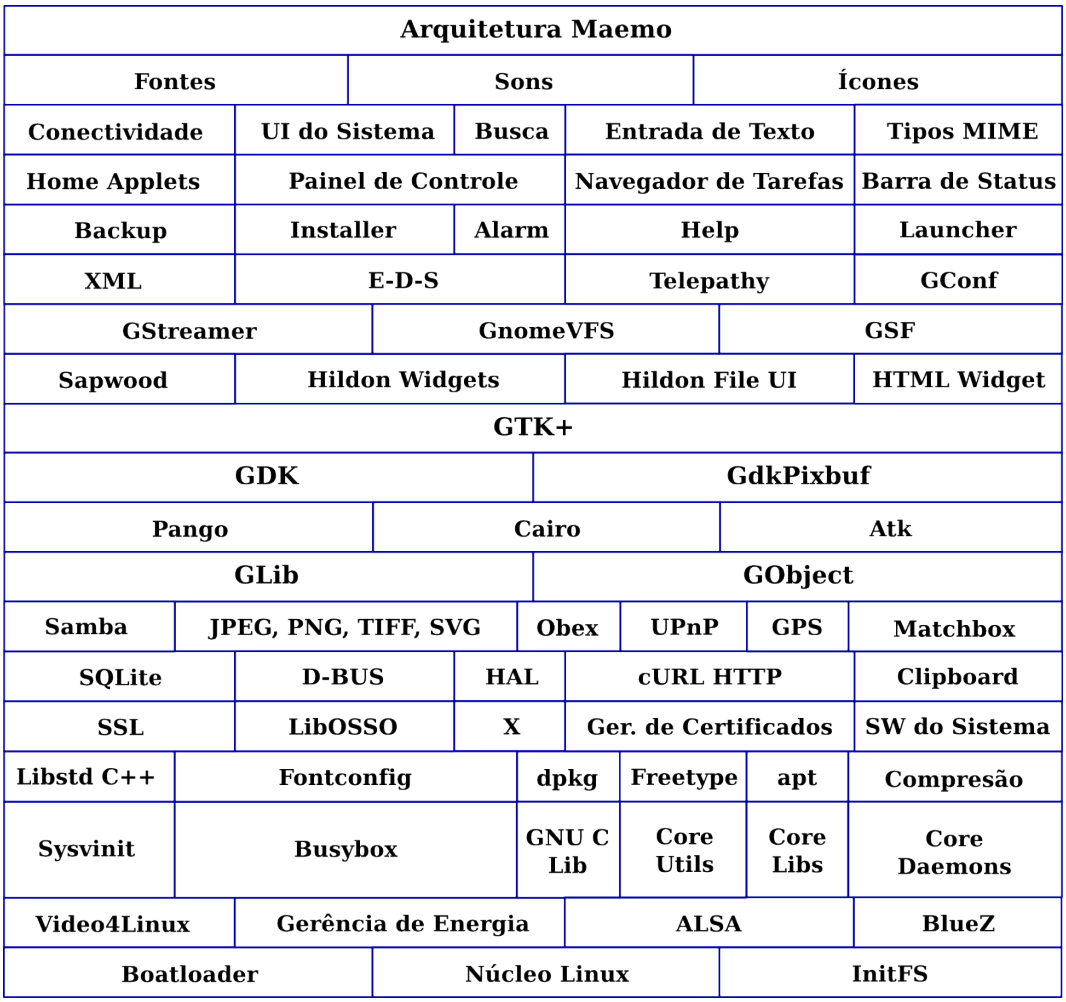

Figura 2.6: Outros componentes que fazem parte da plataforma maemo.

2.5 Internet Tablets

Internet Tablet é uma linha de produtos produzidos pela Nokia. Na Figura 2.7, apresenta-se os quatro diferentes Internet Tablets. São dispositivos leves, menores do que um laptop e maiores do que um PDA. O primeiro produto dessa linha foi o Nokia 770, lançado em 2005. Baseado em uma plataforma OMAP 1710, o Nokia 770 representou um avanço em termos de adoção do sistema operacional Linux para dispositivos embarcados. Em 2006, foi lançado o Nokia N800, que trouxe melhorias em termos de hardware que melhoraram o seu desempenho. No final de 2007, foi lançado o Nokia N810. Ambos, N800 e N810, são baseados na platforma OMAP 2410.

Alguns deles (Nokia N810) possuem um pequeno teclado e todos possuem tela sensível ao toque. Os Internet Tablets foram os primeiros dispositivos portáteis da Nokia que levaram a plataforma Linux embarcada para os usuários finais. A aplicação deve ser desenvolvida levando-se em consideração que o usuário utiliza uma caneta para interagir com ambiente. Há também a possibilidade de utilizar um teclado virtual com a caneta ou um sistema de reconhecimento de escrita. Um conjunto de botões físicos também podem ser utilizados.

Na Tabela 2.2, ilustra-se a comparação entre os três dispositivos. As aplicações devem ser compiladas tendo como alvo o processador ARMv5. Os detalhes do processo de desenvolvimento são descritos no Capítulo 4.

2.6 Conclusão

A plataforma maemo é o resultado do esforço da Nokia em gerar um produto baseado em Linux e outras tecnologias *open source*. Entre muitas publicações especializadas em Linux, o Internet Tablet foi considerado o dispositivo do ano 2006 pela *Linux Journal*. Tal fato mostra o quanto a plataforma foi bem aceita pelos profissionais e também pela comunidade. Trata-se de uma plataforma completa que oferece ao desenvolvedor bibliotecas úteis na construção de aplicações. A plataforma maemo continua em constante expansão, atraindo cada vez mais desenvolvedores para sua comunidade, gerando aplicações inovadoras e agregando valor final aos dispositivos da linha Internet Tablet.

Tabela 2.2: Comparativo entre os diferentes modelos de Internet Tablets.

	Nokia 770	**Nokia N800**	**Nokia N810**
Plataforma	TI OMAP 1710	TI OMAP 2420	TI OMAP 2420
Memória (Flash)	128 MB + 64 MB (RAM)	256 MB + 128 MB (RAM)	256 MB + 128 MB (RAM)
Conectividade	WLAN 802.11 b/g Bluetooth 1.2 USB1.1 com conector Mini B	WLAN 802.11 b/g Bluetooth 2.0 EDR USB2.0 de alta velocidade com conector Mini B	WLAN 802.11 b/g Bluetooth 2.0 EDR USB2.0 de alta velocidade OTG com conector Mini A/B
Armazenamento	1 Leitor de cartão de memória compatível com RS-MMC	2 Leitores de cartão de memória (compatíveis com cartões SD, miniSD, microSD, MMC, e RS-MMC até 8 GB)	2 GB de memória e 1 leitor de cartão de memória (compatível com cartões miniSD e microSD até 8 GB)
GPS Integrado	Não	Não	Sim
Camêra	Não	Sim	Sim
Edição do OS	Internet Tablet OS 2006	Internet Tablet OS 2007 & 2008	Internet Tablet OS 2008
Duração Bateria	Standby: 7 dias Em utilização: 3 horas	Standby: 12 dias Em utilização: 4 horas	Standby: 12 dias Em utilização: 4 horas

Figura 2.7: Os quatro modelos existentes de Internet Tablets: Nokia 770, Nokia N800, Nokia N810 e Nokia N810 WiMax Edition.

Capítulo 3

Ambiente de Execução maemo

Embora a plataforma maemo seja uma distribuição Linux, algumas limitações inerentes à grande parte dos sistemas embarcados, como memória limitada, devem ser consideradas em seu design. Dessa forma, a plataforma maemo possui uma série de componentes que gerenciam de maneira otimizada a inicialização, execução e encerramento de aplicações, bem como os estados da própria plataforma. Neste capítulo, o ambiente de execução maemo é descrito: os elementos que gerenciam o ciclo de vida de uma aplicação maemo e os estados do dispositivo.

3.1 Introdução

O ambiente de execução da plataforma maemo possui componentes importantes utilizados durante o ciclo de vida de uma aplicação, como, por exemplo, o Navegador de Tarefas e o barramento de sessão D-Bus. Tal ambiente, além de ser responsável pela correta execução de um aplicativo maemo, também carrega e finaliza as aplicações, além de implementar algumas abordagens que tornam mais eficiente a utilização da memória. A seguir, estão enumeradas funcionalidades utilizadas durante a utilização do dispositivo:

Carregamento de aplicações O usuário inicializa a aplicação através do Navegador de Tarefas, ou através da Barra de Status (por exemplo, o gerenciador de conexões), ou a partir do Gerenciador de Arquivos (para visualizar um arquivo) ou mesmo através de uma outra aplicação (por exemplo, a funcionalidade “Enviar como E-mail” de um editor de textos);

Gerência de aplicativos Para a correta utilização da memória, somente uma única instância da aplicação pode ser executada ao mesmo tempo. Se a aplicação já estiver em execução, ela receberá uma mensagem sobre uma nova invocação (por exemplo, "open file test.txt") e a executa. O usuário pode escolher qual das as várias aplicações em execução deve preencher a área principal do desktop;

Gerência de memória Como o dispositivo não possui memória suficiente para executar muitas aplicações ao mesmo tempo, o sistema pode encerrar alguma aplicação em *background* que já havia sido considerada "apta" para ser removida;

Finalização de aplicações Aplicações terminam quando usuários as fecham através da interface gráfica ou quando há uma requisição do sistema.

3.2 Componentes

Os componentes envolvidos na gerência do ciclo de vida e troca de aplicação são o Navegador de Tarefas, o barramento de sessão D-Bus, carregador maemo e gerenciador de janelas.

3.2.1 Navegador de Tarefas

O Navegador de Tarefas é uma aplicação da plataforma maemo com funções bastante importantes, por exemplo, carregamento de aplicativos. Para o usuário final, o Navegador de Tarefas é a barra lateral que lista todos os aplicativos, conforme ilustrado na Figura 3.1. Além disso, o Navegador de Tarefas também é usado para:

- Listar todas as aplicações disponíveis no dispositivo para que o usuário selecione a desejada, conforme Figura 3.2;
- Carregar as aplicações. Essa funcionalidade será detalhada mais adiante;
- Encerrar em background as aplicações que estão aptas quando o sistema indica que há pouca memória disponível;
- Listar todas as aplicações em execução e também aquelas que foram encerradas em background, conforme Figura 3.3. Estas últimas aparecem ao usuário como se ainda estivessem em execução;

Figura 3.1: Navegador de Tarefas e outros elementos presentes na tela principal.

- Intercalar aplicações que já estão sendo executadas e aquelas que foram encerradas em background. Isso é feito através de:
  - Envio de uma requisição para o gerenciador de janelas para ativar a janela de uma aplicação;
  - Envio de uma mensagem à aplicação para ativar uma determina visão de janela;
  - Inicialização da aplicação caso ela tenha sido encerrada em background.

3.2.2 Barramento de Sessão D-Bus

Toda aplicação possui um nome conhecido que a identifica. Este nome é único e é utilizado para carregar a aplicação através do barramento D-Bus. Há um serviço D-Bus para cada aplicação.

Aplicações são executadas (ativadas) pelo gerenciador de barramento D-Bus, mais precisamente pelo barramento de sessão. Se não estiver em execução, a aplicação é implicitamente ativada quando uma mensagem de auto-ativação é enviada para o serviço correspondente. O D-Bus obtém o nome do executável (que pode ser um binário ou um script) a partir do arquivo .service relativo à aplicação. O executável é então invocado para inicializar a aplicação. A mensagem de ativação pode também conter parâmetros para a aplicação, por

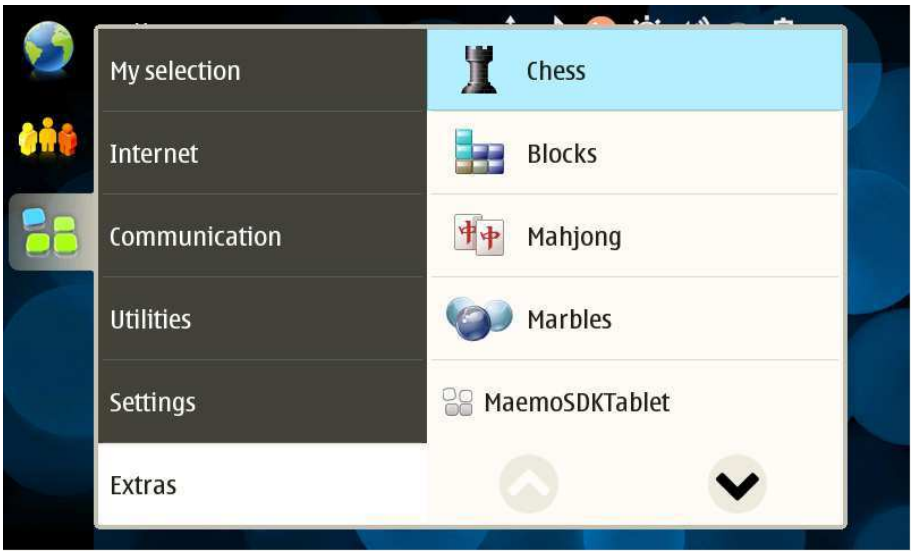

Figura 3.2: Lista de aplicativos disponíveis.

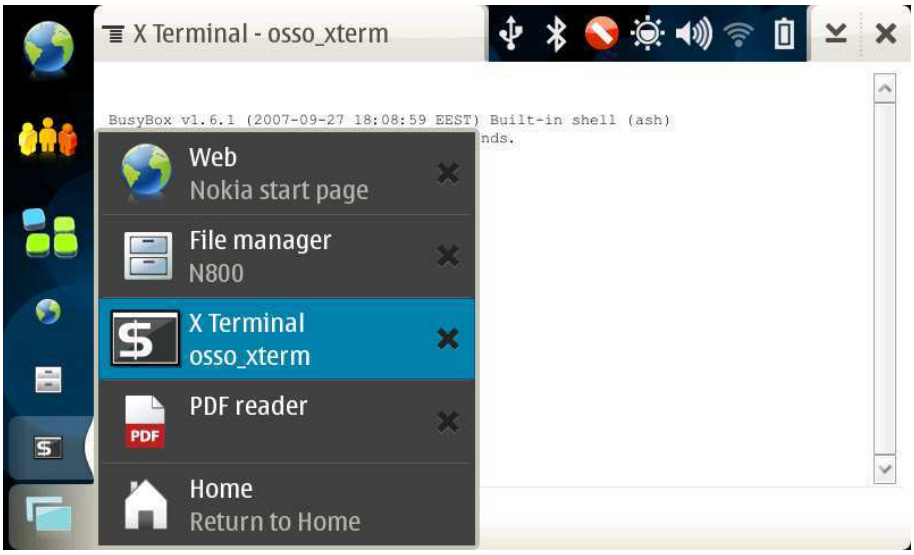

Figura 3.3: Seletor de aplicativos em execução do Navegador de Tarefas.

exemplo, o nome do arquivo que ela deve abrir ao ser iniciada. A ativação via D-Bus garante que somente uma instância da aplicação está sendo executada por vez.

Se o serviço não se registrar no sistema dentro do tempo esperado, o D-Bus assume que a ativação do processo falhou e então encerra o processo inicializado.

3.2.3 Carregador maemo

O carregador maemo é utilizado para acelerar a inicialização da aplicação e também para compartilhar dados inicializados. Ele é composto por duas partes: *maemo-invoker* e *maemo-launcher*. O *maemo-invoker* é executado pelo gerenciador D-Bus ou por scripts para inicializar um dado serviço (aplicação). O uso do *maemo-launcher* requer que a aplicação seja compilada como uma biblioteca compartilhada (*shared library*). Há algumas macros úteis presentes nas ferramentas de empacotamento Debian que fazem com que a aplicação utilize automaticamente o *maemo-invoker*. Assim, o nome executável da aplicação é ligado ao *maemo-invoker* e o nome do executável da aplicação (biblioteca) passar a ter a extensão .launch. O *maemo-invoker* deve esperar até que o *maemo-launcher* avise que a aplicação foi finalizada e então um valor de retorno correto pode ser enviado para quem invocou o serviço.

O *maemo-launcher* é um servidor que inicializa grande parte dos dados utilizados pelas aplicações, como tipos GLib, temas GTK+ e algumas classes de *widgets* GTK+. Quando o *maemo-invoker* solicita que o aplicativo seja iniciado, ele não será executado; será carregado como uma biblioteca compartilhada (através da função `dlload()`), ramificado (através da chamada ao sistema `fork()`) e então a função `main()` é invocada. Com o `fork()`, os dados inicializados serão compartilhados até que sejam modificados. Se a aplicação for finalizada de maneira anormal, o *maemo-launcher* notifica o sistema, que então repassa a informação de erro para o usuário.

3.2.4 Gerenciador de Janelas

O gerenciador de janelas é o Matchbox, o qual é responsável pela intercalação correta das janelas e armazenamento, ou seja, como organizar corretamente a ordem das janelas existentes. Mais adiante, será descrito o processo de gerência das janelas gráficas.

3.3 Gerenciamento do Estado da Plataforma

O estado do dispositivo é gerenciado por alguns componentes de software do sistema:

Gerenciador do Estado do Dispositivo (*Device System Management Entity* - dsm)

Responsável pela gerência dos estados do dispositivo, incluindo desligado e ligado.

Monitora o estado dos processos críticos para o sistema (como D-Bus, X11 e Gerenciador de Janelas), inicia operações para economizar energia quando inativo, etc.;

Controle de Modo (*Mode Control* - mce) Provê interfaces para controle dos modos do dispositivo, como modo *offline* (desabilita o Bluetooth e WLAN) e vários níveis de mudanças da interface do usuário, como bloqueio da tela sensível ao toque e do teclado, LEDs, etc.;

Gerenciamento de Bateria (*Battery Management* - bme) Responsável por monitoração e mudança da voltagem da bateria, carregamento da bateria e identifica quando o carregador foi conectado ao dispositivo.

Um dos componentes mais importantes (e interessante do ponto de vista do programador) é o D-Bus. O barramento de mensagens do D-Bus é o ponto central da arquitetura da plataforma maemo. Aplicações devem “escutar"mensagens do D-Bus que indicam o estado do dispositivo, como “Bateria Fraca"e “Desligado". Quando recebe uma messagem desse tipo, a aplicação deve, por exemplo, perguntar ao usuário se ele deseja salvar algum arquivo que foi aberto ou realizar ações similares para salvar o estado da aplicação.

3.4 Inicialização do Aplicativo

Quando um usuário inicializa uma aplicação através do Navegador de Tarefas, uma mensagem é enviada para o serviço referente à aplicação através do D-Bus. Essa mensagem deve ter o *flag* de auto-ativação. Aplicações também podem ser inicializadas de maneira implícita por outros aplicativos, através também do envio de mensagens D-Bus: por exemplo, ao clicar em um arquivo com tipo MIME já definido, o gerenciador de arquivos pode invocar a aplicação correta.

Os lugares onde diferentes aplicações podem obter o nome do serviço D-Bus de uma determinada aplicação são:

- Navegador de tarefas - o nome do serviço é especificado no arquivo .desktop da aplicação juntamente com o nome e ícones;

- Gerenciador e Navegador de arquivos - o nome do serviço é recuperado do registro de tipos MIME através da biblioteca libosso-mime;

- Outras aplicações que usam APIs de invocação de serviços. As bibliotecas sabem qual o serviço implementado/registrado pelo D-Bus.

O gerenciador D-Bus busca informações no arquivo .service da aplicação para saber como executar a aplicação antes de enviar a mensagem; por exemplo, é preciso saber onde se encontra o executável. A aplicação que foi carregada pelo gerenciador D-Bus terá apenas uma única instância em execução, pois o D-Bus não permite que o mesmo serviço seja registrado por mais de uma vez. Aplicações não podem verificar se o serviço destino das mensagens D-Bus enviadas estão ou não sendo executadas: elas devem apenas enviar todas as mensagens de auto-ativação para solicitar a execução do destinatário.

Utilizar uma mensagem de auto-ativação nem sempre é desejável. Por exemplo, ao se requisitar que uma certa aplicação seja finalizada, não é necessária a utilização de uma mensagem desse tipo. A aplicação ativada aparecerá automaticamente na tela. O Navegador de Tarefas também suporta a execução direta da aplicação, caso o nome do serviço da aplicação não seja especificado pelo arquivo .desktop. Dessa forma, a aplicação carregada **não** precisa saber nada sobre o sistema D-Bus ou biblioteca LibOSSO. Porém, o sistema não garante que haja apenas uma única instância da aplicação sendo executada.

A aplicação pode utilizar o carregador maemo para agilizar sua execução. Na Figura 3.4 ilustra-se como isso ocorre na prática, bem como quais os processos envolvidos nos diferentes métodos de invocação de aplicações.

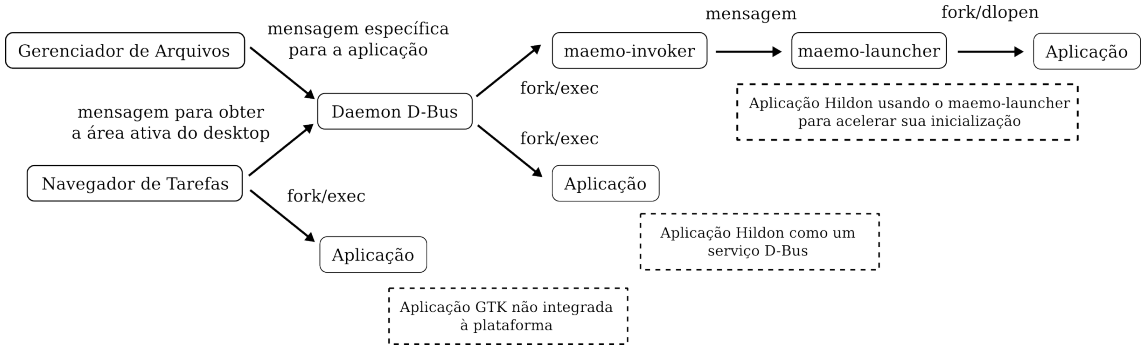

Figura 3.4: Métodos para ativação de aplicações.

3.4.1 Passagem de Variáveis de Ambiente

A aplicação carregada pelo gerenciador D-Bus herda as variáveis de ambiente que foram definidas no momento em que o barramento de sessão D-Bus foi iniciado. D-Bus não provê um outro mecanismo para modificar as variáveis de ambiente passadas para as aplicações quando são ativadas. Se a aplicação for carregada pelo Navegador de Tarefas, pode-se implementar uma função para que ela herde qualquer variável de ambiente. Porém, quando o Navegador de Tarefas utiliza o D-Bus para carregar a aplicação, a situação volta para o cenário anterior: a execução é feita pelo gerenciador D-Bus.

A aplicação carregada pelo *maemo-launcher* herda todas as variáveis de ambiente que foram definidas no momento em que o *maemo-launcher* foi inicializado. Para que as aplicações apresentem o tema de interface gráfica correto, o *maemo-launcher* fica obervando constatemente as mudanças de interface. Uma variável de ambiente não pode ser utilizada para mudanças dinâmicas de configuração, já que é necessário que o aplicativo seja reinicializado para que o novo valor da variável seja definido.

Mudanças de localidade e língua são comunicados através de variáveis de ambiente. Como as aplicações e suas bibliotecas também armazenam o estado da localidade, a modificação das preferências (através de GConf, por exemplo), não ajudaria. Além disso, a mudança da localidade e da língua do dispositivo requer a reinicialização de todas as aplicações e processos envolvidos na invocação da aplicação.

3.5 Finalização do Aplicativo

Uma aplicação é encerrada quando ela é fechada pelo usuário através de botões na interface gráfica, ou quando o sistema solicita tal tarefa. O sistema vai requisitar e forçar que as aplicações sejam finalizadas quando, por exemplo:

- A bateria do dispositivo tenha chegado ao final;
- O usuário desliga o dispositivo;
- O usuário modifica o idioma do dispositivo;
- O usuário modifica as configurações para seguir as definições vindas de fábrica;

- O usuário tenta fechar ou mudar para uma aplicação que não responde e o usuário aceita que o sistema finalize a aplicação.

Caso o sistema esteja com pouca memória disponível, o sistema pode requisitar que as aplicações sejam encerradas em background. Este método será discutido mais adiante. Se não há memória suficiente para que o sistema inicialize a aplicação solicitada, a aplicação vai requisitar mais memória e o núcleo vai encerrá-la devido à falta do recurso.

3.5.1 Armazenamento do Estado e Encerramento em Background

O armazenamento do estado da aplicação é necessário para que o estado da interface gráfica do aplicativo e outros dados do usuário possam ser recuperados, caso a aplicação encerre de maneira anormal ou se seja encerrada em background. As aplicações devem salvar o seu estado constantemente ao serem deixadas em background.

3.5.2 Encerramento em Background

A plataforma maemo possui um mecanismo para encerrar em background aplicações com interface gráfica. Tal procedimento visa economizar memória para que outras aplicações também possam ser executadas. Este processo é chamado de encerramento em background (*background killing*).

O encerramento em background é implementado pelo Navegador de Tarefas, para encerrar uma aplicação de maneira transparente sem que o usuário perceba e para reinicializá-la quando o usuário necessitar da mesma. Isto é possível, pois as aplicações salvam os estados da interface gráfica periodicamente e o Navegador de Tarefas também sabe quais são as aplicações gráficas em execução. As aplicações salvam o estado da interface gráfica sempre que forem para background, ou seja, quando estão atrás de uma janela de outra aplicação.

Contudo, nem sempre é possível salvar o estado da interface gráfica do usuário (por exemplo, quando há um download em progresso). Somente as aplicações que salvaram o estado de sua interface gráfica podem ser encerradas. Quando o sistema notificar que há pouca memória disponível, o Navegador de Tarefas encerrará todas as aplicações que estão aptas. Quando a aplicação for inicializada novamente, será necessário que toda a interface gráfica seja refeita de acordo com o estado que foi salvo anteriormente. O Navegador de

Tarefas não tentará reiniciar a aplicação se não houver memória suficiente no sistema para tal. Há alguns problemas relativos ao encerramento em background na plataforma maemo:

- Suponha que a aplicação que foi encerrada se encontra logo depois de uma aplicação que está na área ativa do desktop. Caso esta última seja finalizada pelo usuário, a aplicação que está logo depois não será visível para o usuário até que tenha sido reiniciada e voltado ao seu estado anterior;

- O Navegador de Tarefas verifica se a nova aplicação pode ser inicializada comparando a quantidade de memória livre disponível no sistema com o valor configurado em tempo de compilação. Tal valor pode ser menor do que uma certa aplicação pode precisar ou, em alguns casos, mais do que o necessário. A quantidade mínima especificada de memória livre desejável é um valor opcional.

3.5.3 Armazenamento do Estado da Interface Gráfica

A biblioteca LibOSSO é a responsável por armazenar o estado da aplicação gráfica do usuário. O LibOSSO salva estes dados em um descritor para cada aplicação. O LibOSSO também assegura que o arquivo de estado não será modificado até que a escrita do arquivo tenha sido completada e o arquivo seja fechado. A aplicação usa a API padrão POSIX que realiza as operações de leitura e escrita nos arquivos.

Se o dispositivo for reiniciado, os estados da interface gráfica das aplicações são descartados. Dessa forma, as aplicações são iniciadas a partir do estado padrão. Cada aplicação com um número de versão diferente (por exemplo, é possível a existência, no mesmo sistema, de duas versões distintas da mesma ferramenta) terá seu próprio arquivo de estado da interface gráfica.

3.5.4 Armazenamento dos Dados do Usuário

Algumas aplicações devem salvar constantemente os dados do usuário quando estão na área ativa do desktop, para que não haja grandes perdas caso ocorra alguma falha, por exemplo, problemas na bateria. As aplicações devem registrar uma função de chamada LibOSSO para realizar o armazenamento das informações e notificar ao LibOSSO quando os dados do

usuário forem modificados (a aplicação se encontra em “dirty state"). Então, o LibOSSO vai notificar às aplicações no momento em que elas devem realizar o armezenamento das informações. Na implementação atual, há somente um temporizador que após X unidades de tempo envia a notificação para armazenamento. Essa abordagem é adotada, pois a tarefa de armazenamento é realizada de maneira síncrona com a gerência de energia do dispositivo.

Quando uma aplicação deixa de ser ativa na área do desktop (ou seja, vai para background), ela deve invocar a função LibOSSO “forced autosave", a qual salva o estado da aplicação. Essa tarefa deve ser realizada pela própria aplicação, pois o LibOSSO não sabe quando ela vai para background.

3.6 Gerenciamento de Janelas

Muitos componentes participam do gerenciamento de janelas. O principal deles é o servidor X, o qual realiza todas as operações de desenho de gráficos e de posicionamento de janelas. O gerenciador de janelas Matchbox implementa a política de gerenciamento de janelas e é responsável por:

- Adequar a barra de títulos e as bordas dos diálogos com o tema atual;
- Movimentar adequadamente os diálogos, pertecentes a uma determinada janela, para o local correto da pilha. Por exemplo, quando uma aplicação passa a ser a principal na área do desktop, os seus diálogos também passam para a área principal;
- Esconder a barra de títulos e painéis (como Barra de Status e Navegador de Tarefas) quando o modo tela cheia for ativado;
- Assegurar que todos os diálogos e janelas do sistema (notificação de falhas, mudança no estado do hardware, etc.) estejam sempre no topo da pilha;
- Assegurar que quando a área de entrada de dados (teclado virtual ou reconhecedor de escrita) for aberta, a janela da aplicação será devidamente redimensionada e os diálogos mudarão de posição para que não se sobreponham à área de entrada de dados;

- Modificar o foco para a janela correta. Por exemplo, quando uma janela é aberta, o foco é dado para a nova janela e quando uma janela for fechada, o foco é dado para a próxima janela.

O Navegador de Tarefas é um outro componente importante para o gerenciamento de janelas. Ele mantém uma lista de todas as janelas e visões abertas das aplicações. A partir dessa lista, o usuário pode selecionar a aplicação a ser exibida na área principal. Se a aplicação tiver sido encerrada em background (o seu estado está salvo), o Navegador de Tarefas continuará a mostrá-la na lista de aplicativos para que pareça que ainda está em execução.

3.6.1 Empilhamento de Aplicativos

A aplicação pode assumir a área principal por si própria, ou o Navegador de Tarefas pode realizar tal tarefa, através do envio de uma mensagem padrão do protocolo X.

Quando uma outra aplicação envia uma mensagem para requisitar algum serviço de uma outra aplicação (por exemplo, o Gerenciador de Arquivos pede que o Visualisador de Imagens mostre um arquivo), ela é automaticamente inicializada (mas não se torna a janela ativa do desktop) pelo gerenciador D-Bus. Quando a aplicação recebe a mensagem, sua janela pode ser enviada para o topo se houver a necessidade de interação com o usuário.

3.6.2 Registro de Janelas no Navegador de Tarefas

Para que o Navegador de Tarefas coloque uma determina janela da aplicação no topo da pilha, é necessário identificar quais janelas pertencem a cada aplicação. Para isso, utilizamos o arquivo `.desktop` da aplicação e, mais precisamente, o campo StartupWMClass.

Quando uma aplicação é carregada pelo gerenciador D-Bus, o Navegador de Tarefas compara a propriedade WM_CLASS da janela da aplicação que está aberta com o campo StartupWMClass, presente no arquivo .desktop da aplicação. Dessa forma, é possível identificar a qual aplicação a janela pertence. O GTK modifica automaticamente a propriedade WM_CLASS da janela para o nome do executável. Os ícones da aplicação que são mostrados na lista de aplicativos e o seletor de aplicativos do Navegador de Tarefas também são obtidos a partir do arquivo `.desktop`.

Se a propriedade WM_CLASS da janela em questão não for encontrada em nenhum arquivo .desktop (no campo StartupWMClass), não haverá nenhum ícone para a aplicação no seletor de aplicativos do Navegador de Tarefas e também o usuário não poderá voltar para a aplicação.

3.6.3 Encerramento das Janelas de Aplicações

As janelas das aplicação são armazenadas em uma pilha. Quando uma aplicação assume a área principal, então sua janela é alternada para o topo da pilha. Quando a janela é fechada, a próxima janela existente na pilha passa a assumir o desktop e então fica no topo. A pilha de janelas das aplicações é separada da pilha de janelas do sistema (diálogo).

3.7 Problemas no Comportamento de Aplicações

O framework de aplicação assume que os aplicativos se comportam de maneira esperada. Entretanto, em algumas situações, ele verifica se a aplicação está paralizada através do envio de mensagens _NET_WM_PING do padrão EWMH (*Extended Window Manager Hints*) e, logo após, fica esperando uma resposta da aplicação. Por exemplo, uma aplicação GTK+ responderá automaticamente a mensagem caso esteja executando o laço principal de espera de eventos. A verificação é realizada pelo gerenciador de janelas Matchbox toda vez que:

- O botão para fechar a janela da aplicação for pressionado, ou seja, o usuário tenta fechar a aplicação;
- O Navegador de Tarefas solicitar que uma outra janela de aplicação assuma a área principal, ou seja, o usuário tenta selecionar uma aplicação a partir do alternador de aplicativos do Navegador de Tarefas. Isto é necessário, pois o botão de fechar a janela não é visível para aplicações em modo tela cheia.

Se a aplicação não responder no tempo esperado, o gerenciador de janela informa ao Navegador de Tarefas, o qual apresentará ao usuário um diálogo perguntando se a aplicação pode ser finalizada. Se o usuário aceitar, então os signais TERM e KILL são enviados em seqüência para o processo da aplicação (o PID do processo é obtido da propriedade _NET_WM_PID da janela) e o encerramento é gravado em um log pelo sistema. O diálogo

é removido automaticamente caso a aplicação responda enquanto o diálogo estiver aberto. Se a aplicação maemo terminar de maneira anormal, o maemo-launcher envia uma mensagem ao Desktop para informar o usuário sobre o problema.

3.8 Conclusão

Sistemas embarcados possuem uma séria de limitações que devem ser consideradas pelos sistemas que executam em tais plataformas. Gerência correta da (pouca) memória disponível, otimização da execução de aplicativos e gerenciamento do consumo da bateria são exemplos de alguns problemas presentes que o ambiente de execução maemo resolve com seus vários componentes: Navegador de Tarefas, barramento da sessão D-Bus, dentre outros. Alguns desses elementos descritos nesse capítulo, como o sistema D-Bus, serão tratados em capítulos posteriores.

Capítulo 4

Desenvolvimento de Aplicações

O processo moderno de desenvolvimento de software engloba diversas tarefas realizadas iterativamente, tais como projeto arquitetural, documentação, codificação, teste, integração, etc. No desenvolvimento de software para sistemas embarcados, além das tarefas comuns, outros detalhes devem ser considerados, por exemplo, execução no dispositivo alvo e gerência correta de recursos. Neste capítulo, descreve-se o processo e as práticas comuns do desenvolvimento de aplicações para a plataforma maemo.

4.1 Introdução

A comercialização dos dispositivos embarcados contribui para a disseminação da computação pervasiva. Esta contribuição será certamente mais significativa para aqueles dispositivos que são baseados em sistemas operacionais abertos e de código livre, por exemplo, os que utilizam Linux Embarcado como sistema operacional. O desenvolvimento de aplicativos que exploram as características de conectividade, mobilidade e interatividade dos sistemas embarcados são os vetores que aceleram a utilização de computação pervasiva.

Geralmente, aplicações para sistemas embarcados são desenvolvidas em máquinas *desktop*, com um poder computacional maior do que dos sistemas embarcados. Com as restrições presentes em plataformas embarcadas, tarefas como a compilação da aplicação consumiria muito tempo caso fossem realizadas no próprio dispositivo [91; 92; 41].

Contudo, embora o desenvolvimento ocorra em um ambiente diferente do sistema em-

barcado, o produto final será executado no ambiente embarcado alvo, considerando todas as restrições presentes. Assim, boas ferramentas para o desenvolvimento de aplicações para Linux embarcado devem considerar este aspecto e buscar simular em máquinas *desktop*, as restrições do ambiente alvo.

4.2 Ambiente de Desenvolvimento

O ambiente usado no desenvolvimento de aplicações maemo é chamado de `maemo SDK` [1], disponível gratuitamente para a comunidade através do site [58]. O maemo SDK utiliza o Scratchbox [65], o qual é um conjunto de ferramentas para o desenvolvimento de aplicações para Linux embarcado. A instalação do ambiente pode ser realizada através de três métodos: i) manual; ii) automático, utilizando scripts de instalação que fazem o download dos pacotes necessários e configuram todo o ambiente; e iii) automático, via gerenciador de pacotes.

O maemo SDK provê um ambiente de desenvolvimento utilizado na criação de software para os Internet Tablets utilizando um ambiente *desktop*. O SDK executa dentro do ambiente Scratchbox e contém todos os compiladores, ferramentas, bibliotecas e arquivos cabeçalhos para desenvolver aplicações para as duas arquiteturas alvo: Intel (x86) e ARMEL. A Figura 11.1 ilustra o ambiente de desenvolvimento maemo baseado em linha de comando.

O desenvolvimento da aplicação e testes preliminares são realizados no ambiente x86, o qual também possui o desktop Hildon para executar o software no computador desktop (usando um servidor X virtual, como o Xephyr [30]) com a mesma interface que serão executadas no Internet Tablet. Na Figura 4.2, estão ilustrados o servidor X virtual e o desktop Hildon em execução. Ao utilizar o computador desktop para desenvolvimento, obtem-se um processo bastante similar ao desenvolvimento de aplicações Linux normais. O maemo SDK também possui suporte para *plug-ins* Eclipse, um ambiente de desenvolvimento integrado (*Integrated Development Environment* - IDE) que melhora a produtividade, ajudando o programador no processo de desenvolvimento. Maiores detalhes sobre os ambientes disponíveis são descritos no Capítulo 12.

Ao finalizar os testes preliminares no ambiente desktop, a próxima fase é compilar a aplicação e empacotá-la para a arquitetura ARMEL usando o alvo ARMEL disponível no

[1] maemo SDK Releases. Disponível em `http://maemo.org/development/sdks/`.

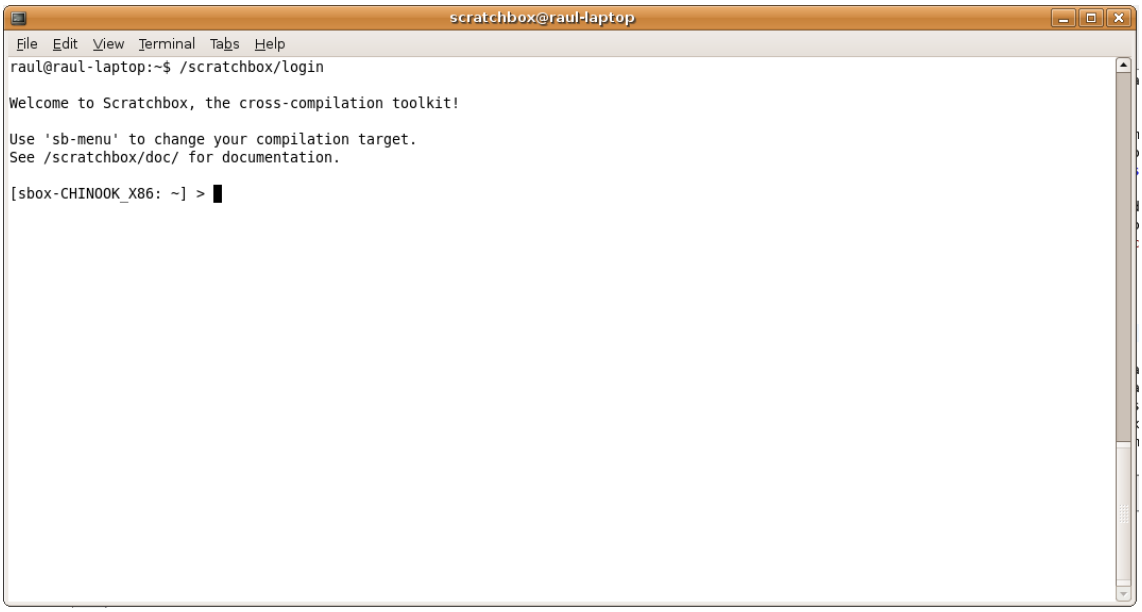

Figura 4.1: Scratchbox e maemo SDK instalados.

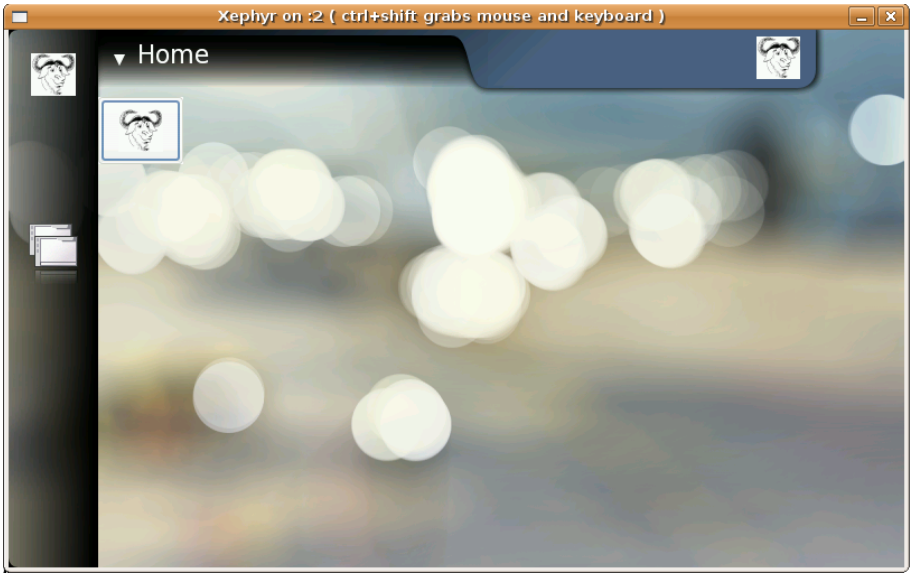

Figura 4.2: Framework de Aplicações Hildon inicializado.

Scratchbox. Dessa forma, a aplicação já se encontra pronta para ser executada e testada no Internet Tablet. A fase de testes realizada no próprio Internet Tablet é bastante importante mesmo que a aplicação seja executada no ambiente desktop corretamente: o maemo SDK não é 100% idêntico ao dispositivo. A seguir estão listadas as ferramentas de desenvolvimento e recursos usados no processo de desenvolvimento de aplicações maemo:

Scratchbox Ambiente de compilação cruzada desenvolvida para facilitar o desenvolvimento de aplicações Linux embarcadas. Também provê um conjunto de ferramentas para integrar e compilar até mesmo toda uma distribuição Linux completa para uma plataforma embarcada. O Scratchbox permite o desenvolvimento de aplicativos para as arquiteturas ARM e x86, além de PowerPC e MIPS, por exemplo;

maemo SDK rootstraps Um rootstrap é uma imagem de um sistema de arquivos para um determinado alvo usado pelo Scratchbox como base do desenvolvimento. Maemo SDK provê rootstraps para os ambientes ARMEL e x86;

Binários Nokia São pacotes de software que não possuem código fonte disponível, mas disponibilizam uma API pública. Como exemplo, existem as bibliotecas de GPS (localização) e importação/exportação de contatos;

Ferramentas maemo O maemo SDK também disponibiliza recursos para programadores que precisam de ferramentas mais sofisticadas: análise de código, depuração, testes automatizados, perfilação, etc.;

Repositórios O site maemo.org possui diferentes repositórios que podem ser utilizados através das ferramentas de instalação baseadas no padrão Debian;

Documentação Documentação para o desenvolvimento de software para a plataforma maemo: How-Tos, tutoriais, referências de API, manuais de ferramentas e outros vários guias. Disponível no *site* maemo.org;

Exemplos Os pacotes de exemplos incluem código fonte demonstrativo, os quais utilizam diferentes APIs que podem ser obtidos dos repositórios maemo.

A seguir estão listadas as fases do processo de desenvolvimento de aplicações maemo usando a linguagem C/C++:

1. Criar um projeto (possivelmente utilizando exemplos) no Scratchbox para a aplicação;
2. Criar ou atualizar o código fonte e recursos necessários;
3. Criar ou atualizar o arquivo de descrição da interface gráfica (possivelmente utilizando um construtor de interface gráfica);

4. Compilar a aplicação no ambiente x86;
5. Executar e testar a aplicação no ambiente x86;
6. Depurar a aplicação no ambiente x86;
7. Compilar a aplicação no ambiente ARMEL;
8. Executar e testar a aplicação no Internet Tablet;
9. Depurar a aplicação no Internet Tablet;
10. Criar um pacote de instalação ARMEL para o Internet Tablet;
11. Instalar o pacote no Internet Tablet.

A seguir, enumeramos as fases do processo de desenvolvimento de aplicações maemo usando a linguagem Python:

1. Criar um projeto (possivelmente utilizando exemplos) no Scratchbox para a aplicação;
2. Criar e atualizar o código fonte e recursos necessários;
3. Carregar e testar a aplicação usando o ambiente x86;
4. Depurar a aplicação usando o ambiente x86;
5. Executar e testar a aplicação no Internet Tablet;
6. Depurar a aplicação no Internet Tablet;
7. Criar um pacote de instalação ARMEL para o Internet Tablet;
8. Instalar o pacote no Internet Tablet.

Os capítulos seguintes mostrarão mais detalhes sobre cada uma destas fases do processo de desenvolvimento.

4.2.1 Criação de Projetos para a Aplicação

A criação de projetos para aplicações pode ser realizada do início ou utilizando um conjunto de exemplos disponíveis no *site* `maemo.org`. O código fonte pode ser editado usando o editor preferido do programador. O Scratchbox cria um ambiente isolado (também chamado de *sandbox*), um sistema de arquivos separado (chamado *rootstrap*) dentro do sistema de arquivos do ambiente (*desktop* Linux), além de links simbólicos para diretórios dentro do Scratchbox para facilitar o acesso de arquivos a partir do ambiente desktop. É possível também utilizar um editor de texto simples (como vi) dentro do Scratchbox.

Como a plataforma maemo utiliza um sistema de gerência de pacotes baseado na distribuição Debian, é uma boa prática criar a estrutura necessária para criação de pacotes no projeto de aplicativo ou utilizar alguma ferramenta (GNU Autotools) que cria de maneira automática os arquivos e pastas necessários. Dessa forma, a aplicação pode ser facilmente "empacotada"e distribuída para os usuários finais. O desktop Hildon também deve estar inicializado antes de executar aplicações maemo dentro do SDK.

Há alguns ambientes para construção de interfaces gráficas que tornam a criação de aplicativos mais ágil, por exemplo, o Gazpacho e Glade. Um guia completo para instalação do ambiente de desenvolvimento pode ser encontrado no *site* maemo.org.

4.2.2 Compilando e executando as aplicações

Com a criação do projeto e o código fonte em mãos, a aplicação está pronta para ser compilada e testada. O SDK provê todas as ferramentas de desenvolvimento Linux habituais dentro do Scratchbox e também o *framework* de aplicação Hildon. Assim, as aplicações possuem o mesmo comportamento e interface gráfica que teriam no Internet Tablet.

A seguir, listamos as ferramentas de desenvolvimento mais utilizadas pelo maemo rootstrap:

GNU toolchain Termo genérico utilizado para as ferramentas de programação presentes no projeto GNU, que são:

GCC (GNU Compiler Collection) [4; 36] Compiladores e linkers para C, C++, etc.;

GNU Autotools [45] Conjunto de ferramentas de programação úteis para auxiliar a geração de arquivos Makefile e questões relativas à portabilidade;

GNU Make [62] Make é uma ferramenta que controla a geração de executáveis e de outros arquivos genéricos a partir do código fonte do programa;

GNU Binutils [45] Conjunto de ferramentas de programação para manipulação de código objeto em vários formatos de arquivos objeto;

pkg-config [29] Ferramenta usada durante a compilação de aplicações e bibliotecas que auxiliam o programador a inserir as opções corretas para o compilador encontrar as bibliotecas necessárias.

Ferramentas de empacotamento Debian Utilizadas para criar pacotes Debian.

O processo de compilação e execução é detalhado a seguir:

- Compilar a aplicação usando o ambiente x86;
- Executar a aplicação no ambiente x86. Há um *shell script* útil no rootstrap maemo chamado `run-standalone.sh` que deve ser utilizado durante a execução de aplicativos no Scratchbox. O script configura corretamente o ambiente para que a aplicação utilize o framework de aplicações Hildon;
- Testar a aplicação;
- Se necessário, modificar o código e repetir o processo.

A depuração no ambiente x86 é também fácil:

- Use o ambiente x86 no Scratchbox;
- Carregue a aplicação no depurador;
- Execute e depure a aplicação;
- Modifique e compile a aplicação novamente caso necessário.

Aplicações podem ser depuradas no ambiente x86 utilizando as mesmas ferramentas de desenvolvimento e depuração Linux, tais como GNU Debugger (gdb) [75; 60], valgrind [68], ltrace e strace. Algumas ferramentas oferecem interface gráfica para depuração.

A ferramenta valgrind não está disponível para o ambiente ARMEL, somente no x86. Valgrind é uma ferramenta bastante útil para a detecção de vazamento de memória, perfilamento, etc. Se alguma ferramenta de depuração estiver ausente no ambiente de desenvolvimento, é possível adicioná-la no Scratchbox. Para isso, é necessário apenas compilar o código fonte dentro do ambiente, já que o Scratchbox é praticamente um sistema Linux completo. Outra possibilidade é executar a aplicação diretamente no Internet Tablet e depurá-la remotamente utilizando o `gdbserver` no dispositivo e o `gdb` no computador *desktop*, através de uma conexão via cabo ou sem fio. Maiores detalhes sobre depuração são explorados no Capítulo 11.

Compilação Cruzada para Arquitetura ARMEL

A compilação cruzada de aplicativos para os Internet Tablets é simples: basta ativar o ambiente ARMEL no Scratchbox e recompilar a aplicação. Não há diferença alguma no processo de compilação de programas nos ambientes ARMEL e x86. Após a compilação, o executável gerado está pronto para ser executado nos Internet Tablets.

O ambiente ARMEL deve ser utilizado somente para a compilação cruzada e empacotamente de aplicativos para os dispositivos. Como o emulador qemu [12; 10] para arquitetura ARM pode não prover o comportamento esperado de execução, os testes somente no ambiente *desktop* não são suficientes.

Executando, Testando e Depurando Aplicações no Internet Tablet

Embora o maemo SDK seja bastante parecido com o ambiente do Internet Tablet, deve-se considerar que não são totalmente idênticos. Especialmente se a aplicação estiver utilizando um recurso de hardware específico do Internet Tablet, o aplicativo pode se comportar de maneira diferente no SDK e no dispositivo. Felizmente, o executável pode ser testado no Internet Tablet de maneira transparente no Scratchbox, utilizando SSH [9] ou a ferramenta de transparência de arquitetura chamada sbrsh (Scratchbox Remote Shell) [64]. A conexão com o dispositivo é estabelecida através de um cabo USB ou sem fio (wireless). Também é possível executar a aplicação no dispositivo através de conexões usando as ferramentas SCP [9] (para cópia do executável) e SSH (para execução do aplicativo).

Transparência de arquitetura é uma técnica que permite a cópia dos binários gerados no

ambiente ARMEL a partir do computador *desktop* Linux, através de uma conexão, para o dispositivo onde são executados nativamente no Internet Tablet usando o processador ARM. Esta é a melhor maneira de testar a aplicação gerada no ambiente ARMEL. Outra forma seria a utilização do emulador QEMU no ambiente Linux *desktop*. Contudo, o emulador não provê o mesmo comportamento do dispositivo baseado na arquitetura ARM. Dessa forma, a técnica de transparência de arquitetura no dispositivo é uma maneira conveniente para dirigir a bateria de testes do aplicativo desenvolvido no ambiente ARMEL. Ainda é necessário um ambiente gráfico para executar as aplicações gráficas maemo, pois a resolução de tela do Internet Tablet é diferente (800x480) e o tamanho de tela é menor (4.1 polegadas). Dessa forma, utiliza-se um servidor X virtual, por exemplo, Xephyr e Xnest. Assim, realiza-se os seguintes passos para executar e testar a aplicação no dispositivo:

- Copiar o executável para o Internet Tablet (caso esteja utilizando transparência de arquitetura, não é necessário este passo);
- Executar a aplicação no Internet Tablet;
- Testar a aplicação;
- Se necessário, modificar e compilar o código novamente, repetindo o processo.

É possível utilizar o depurador gdb remotamente com o gdbserver sendo executado no Internet Tablet. A aplicação a ser depurada deve ser executada no dispositivo, o que permite um resultado totalmente fiel. Assim, é possível a depuração remota da aplicação usando o ambiente *desktop*.

4.2.3 Implantação e Instalação de Aplicações

A plataforma maemo possui um sistema de gerência de pacotes Debian para instalar e gerenciar os pacotes das aplicações e suas dependências. O gerente de pacotes é transparente: a instalação e remoção dos aplicativos são realizados pelo Gerente de Aplicações no Internet Tablet. O sistema de gerência de pacotes Debian utiliza pacotes e é composto por binários de aplicativos, bibliotecas opcionais, dados que descrevem o pacote, dependências para outros pacotes e funções que podem ser realizadas antes e após o processo de instalação. O empacotamento de aplicativos é realizado utilizando as ferramentas Debian.

Após criar um pacote Debian (a criação na plataforma maemo é idêntica ao processo no ambiente Linux desktop), o aplicativo está pronto para ser instalado no Internet Tablet. O pacote pode ser copiado para o dispositivo e instalado usando o Gerente de Aplicações ou hospendando-os em um repositório de pacotes (um site web ou FTP que contém pacotes de aplicativos) e criando um arquivo usado para instalação através de um simples clique. Esta última opção elimina a necessidade do usuário gerenciar os repositórios manualmente através do Gerente de Aplicações, provendo uma maneira fácil para o usuário final instalar as aplicações.

4.3 Suporte a Linguagens de Programação

A plataforma maemo possui suporte nativo para a linguagem C. Grande parte das aplicações para a plataforma são desenvolvidas na linguagem devido à disponibilidade de bibliotecas e documentação disponível das mesmas, o que acaba tornando sua utilização mais fácil. Contudo, há também vínculos (bindings) entre C e as linguagens C++ e Python [55; 54]. Vínculos são métodos de C++ ou Python que, na verdade, invocam funções escritas em C para realizar todo o trabalho. Os vínculos representam uma maneira rápida e segura de implementar bibliotecas já existentes em C para as linguagens C++ e Python.

A linguagem Python vem sendo cada vez mais utilizada na plataforma, pois é uma linguagem simples que prioriza a velocidade de desenvolvimento e a expressividade. Python é expressiva, com abstrações de alto nível. Na grande maioria dos casos, um programa em Python será muito mais curto que seu correspondente escrito em outra linguagem. Isto também faz com que o ciclo de desenvolvimento seja rápido e apresente potencial de defeitos reduzido - menos código, menos oportunidade para errar. Com muitas bibliotecas nativas já portadas para a linguagem, é possível o desenvolvimento de aplicações multimídia sofisticadas, por exemplo, o Canola [70].

PyMaemo [71] é o suporte oferecido pela plataforma maemo à linguagem de programação Python. Desenvolvida desde 2004 pelo Instituto Nokia de Tecnologia (INdT), PyMaemo possui um tamanho menor e também implementa um conjunto de otimizações necessárias para obter melhorias de desempenho para dispositivos baseados na plataforma maemo. Também provê módulos importantes para a plataforma, por exemplo BlueZ,

GStreamer, Hildon e GTK+. PyMaemo possui os seguintes módulos: iPython, PyBlueZ, PyGame, PyGconf, PyGnomeVFS, PyGobject, PyGtk, Pyid3lib, PyImage, Pyrex, Python, Python-dbus, PythonGPS-bt, Python-Gstreamer, Python-Hildon, Python-Numeric, Python-Osso, Python-Xml, Python-ABook, Evolution-python, Galago-python, PyC URL, Cython e Storm.

Há ainda, o suporte a outras linguagens na plataforma: C#, Ruby, Perl e Java. Contudo, poucas bibliotecas do sistema estão portadas para estas linguagens, o que as torna pouco utilizadas no desenvolvimento de aplicações para a plataforma maemo.

4.3.1 Integração da Aplicação com a Plataforma

Em ambientes Linux desktop, é possível integrar aplicações ao menu principal de maneira simples através de uma interface gráfica, adicionando um atalho ou simplesmente copiando a aplicação para a área de trabalho. Contudo, a integração de aplicações ao navegador de tarefas e ao sistema D-Bus na plataforma maemo não é realizada de maneira simples, uma vez que não há auxílio de interface gráfica. Nesta seção, descreve-se como é possível adicionar uma aplicação ao navegador de tarefas e ao conjunto de serviços inscritos na plataforma através do sistema D-Bus.

Para que uma aplicação gráfica apareça no menu do Navegador de Tarefas e seja controlada através dele, é necessária a criação de dois arquivos: um deles é utilizado para que a aplicação seja ativada através do D-Bus (arquivo `service`), enquanto o outro é usado para integrá-lo à estrutura do menu (arquivo `desktop`). Abaixo está o conteúdo do arquivo desktop mais simples possível, o qual deve existir para que o navegador de tarefas carregue a aplicação através do sistema D-Bus. Programas desenvolvidos para a plataforma maemo normalmente são inicializados através do mecanismo de ativação implementado pelo D-Bus, o qual determina claramente qual o nome do serviço (org.maemo.myapp, por exemplo).

```
[Desktop Entry]
Encoding=UTF-8
Version=1.0
Type=Application
Name=MyApp
```

```
Exec=usr/bin/myapp
X-Osso-Service=org.maemo.myapp
Icon=qgn_list_gene_default_app
```

A correta localização do executável é no diretório /usr, senão a aplicação e seus arquivos serão instalados nos locais incorretos e não serão encontrados pelo navegador de tarefas. O arquivo desktop especifica o texto que deve ser exibido nos menus (atributo `Name`), o ícone associado à aplicação (atributo `Icon`) e até mesmo o nome do serviço D-Bus (atributo `X-Osso-Service`) para que a aplicação seja devidamente ativada. Todos os atributos possíveis para o arquivo desktop encontram-se em XXX. É necessário que exista um arquivo desktop para cada aplicação gráfica, devendo ser copiado para o diretório `/usr/share/applications/hildon`.

Uma vez que o servidor D-Bus possa ativar a aplicação sob demanda e tenha certeza de que somente uma instância da aplicação esteja em execução, a aplicação necessitará de um arquivo service instalado, como o exemplo abaixo:

```
[D-BUS Service]
Name=org.maemo.myapp
Exec=usr/bin/myapp
```

O nome do serviço deve ser exatamente igual àquele presente no arquivo desktop e também idêntico ao nome utilizado para registro através da biblioteca LibOSSO (mais adiante). O nome do arquivo também deve seguir o nome do serviço se possível. O arquivo service precisa estar localizado no diretório `/usr/share/dbus-1/services` para que o servidor D-Bus o encontre. Contudo, apenas copiar o arquivo para o diretório correto não é o suficiente para que o servidor perceba que se trata de um novo serviço e possa inicializá-lo.

O atributo `Exec` deve apontar para o diretório absoluto do alvo onde o servidor D-Bus encontrará o executável para iniciar o serviço quando requerido. Se o executável em questão não se registrar com o mesmo nome conforme descrito pelo atributo `Name`, ele será eventualmente encerrado pelo finalizador de aplicativos.

É necessário registar a aplicação como um serviço D-Bus para que ele seja automaticamente encerrado pelo sistema. Realizamos essa tarefa através da função `osso_initialize()`. O primeiro parâmetro é o nome do serviço D-Bus (deve ser o mesmo do nome usado no arquivo service). A versão do software parece ser desnecessária nesse momento, mas a idéia é que seja possível a existência de várias versões diferentes do mesmo programa em execução ao mesmo tempo, sem colidir com o espaço de nomes do D-Bus. Os outros dois parâmetros para aplicações gráficas normais são sempre TRUE e NULL. Veja a documentação LibOSSO para maiores detalhes.

4.4 Conclusão

Lançado em 2005, o maemo SDK foi criado com o intuito de oferecer ao desenvolvedor as ferramentas para o desenvolvimento de aplicações para a plataforma maemo. Muitas funcionalidades foram adicionadas (ferramentas, compiladores, opções para depuração), bem como melhorias significativas no processo de instalação e configuração do ambiente. Atualmente, é possível instalar todo o ambiente de desenvolvimento de maneira automática. Contudo, mesmo com as ferramentas presentes, é necessário muito tempo e esforço por parte do desenvolvedor na gerência e na configuração destas ferramentas para compor o ambiente de desenvolvimento.

O desenvolvedor deve mudar constantemente de ambiente, codificando o software em algum editor de texto de sua preferência (geralmente com interface gráfica) e realizando as demais atividades no Scratchbox. Isto demanda maior tempo e esforço de desenvolvimento, aumentando também a possibilidade de erros no processo. No Capítulo 12, são apresentados alguns ambientes gráficos de desenvolvimento que auxiliam o programador, trazendo ganhos de produtividade, tempo e qualidade do software.

Capítulo 5

Comunicação

Neste capítulo, é apresentada uma discussão sobre os mecanismos existentes no maemo para comunicação entre aplicativos em toda a plataforma. O principal (e mais usado) método de comunicação é feito através da troca de mensagens entre as aplicações. O perfeito gerenciamento de todas as mensagens, bem como a passagem para os destinatários corretos é realizado pelo sistema D-Bus, também utilizado em outros ambientes bastante conhecidos, como o GNOME. O D-Bus já foi citado em outros capítulos (por exemplo, os Capítulos 4 e 2), os quais descrevem outras utilizações do sistema na plataforma maemo.

5.1 Introdução

Na plataforma maemo, as aplicações precisam se comunicar entre si: troca de mensagens, requisição de serviços e também notificação de eventos. Existem diversas maneiras de realizar tais tarefas, por exemplo, troca de mensagens ou até mesmo soluções de mais baixo nível, como memória compartilhada e fila de mensagens. Contudo, para grande parte das aplicações, é necessária uma solução mais simples e direta com o intuito de facilitar o desenvolvimento e manutenção do software. Para isso, a plataforma maemo adotou o sistema D-Bus para realizar a comunicação entre as diversas aplicações e prover um mecanismo simples de notificação de eventos de software e hardware.

5.1.1 D-Bus

D-Bus é um mecanismo de comunicação entre processos (*Inter Process Communication* - IPC) [35] relativamente novo que foi desenvolvido para ser usado como um *middleware* para ambientes desktops. Existem muitas outras tecnologias que têm como propósito oferecer serviços de IPC, por exemplo CORBA, DCE, DCOM, DCOP, XML-RPC, SOAP, MBUS e *Internet Communications Engine* (ICE). Contudo, cada uma delas foram desenvolvidas para tipos específicos de aplicações. O D-Bus foi criado para atender a dois tipos de situações: I) comunicação entre aplicativos de desktop presentes na mesma sessão, integração da sessão do desktop como um todo e gerenciamento do ciclo de vida de processos (desde o início, até a finalização); II) comunicação entre a sessão do desktop e o sistema operacional.

O GNOME e Hildon são exemplos de projetos que utilizam o D-Bus. Se comparado com outros middlewares para IPC, D-Bus não possui uma série de funcionalidades mais complexas e, por isso, é mais simples e rápido.

D-Bus não oferece mecanismos de baixo nível como sockets, memória compartilhada ou troca de mensagens. Cada um desses mecanismos possuem seus próprios usos, os quais normalmente não são substituídos pelo sistema D-Bus. Por outro lado, D-Bus procura prover funcionalidades de mais alto nível, tais como:

- Espaço de nomes estruturado;
- Arquitetura independente de formato de dados;
- Suporte à grande parte dos tipos de dados usados nas mensagens;
- Uma interface genérica de chamada remota com suporte a exceções (erros);
- Uma interface genérica de sinalização para suporte a *broadcasts*;
- Separação clara de escopos a nível de usuário e a nível de sistema, a qual é bastante útil para tratar com sistemas multi-usuários;
- Não é restrito a nenhuma linguagem de programação específica.

O design do D-Bus é fruto de uma longa experiência com o uso de outras soluções de IPC no ambiente desktop, permitindo que o resultado fosse otimizado. O sistema não possui

funcionalidades desnecessárias (*creeping featurism*), ou seja, funções extras somente para satisfazer casos bem específicos. Conforme discutido anteriormente, o D-Bus procura prover uma maneira fácil para comunicação de processos entre aplicações, gráficas ou não.

D-Bus é uma peça fundamental na plataforma maemo: é o mecanismo de comunicação entre processos, adotado para acessar a serviços providos pela plataforma (e dispositivos). Prover serviços utilizando o D-Bus é a maneira mais fácil para assegurar a reutilização de componentes entre aplicações.

Arquitetura D-Bus e Terminologia

No D-Bus, um barramento é um conceito importante. É o canal do qual as aplicações podem receber e enviar sinais, além de invocar métodos. Há dois barramentos pré-definidos: o barramento de sessão e o barramento do sistema.

- O barramento de sessão é utilizado para comunicação entre aplicações que estão conectadas na mesma sessão do desktop e é normalmente inicializada e executada por um usuário (usando o mesmo identificador de usuário, ou UID);
- O barramento do sistema é utilizado para comunicação entre aplicações ou serviços que executam em sessões distintas. O uso mais comum para este barramento é o envio de notificações do sistema quando ocorrem eventos na plataforma, por exemplo, a adição de um novo dispositivo de armazenamento e mudanças na conectividade da rede.

Normalmente, somente um barramento de sistema existe por vez. Pode haver vários barramentos de sessão, um para cada sessão de desktop. No Internet Tablet, todos as aplicações são executadas com o mesmo identificador de usuário (UID). Portanto, haverá somente um barramento de sessão também.

Um barramento existe no sistema sob a forma de um *daemon de barramento*, o qual é responsável por repassar mensagens de um processo para outro. O *daemon* também deve enviar notificações a todas as aplicações do barramento. Em baixo nível, D-Bus suporta somente comunicação do tipo ponto-a-ponto, através do uso de sockets de domínio local (AF_UNIX) entre a aplicação e o daemon de barramento. A comunicação ponto-a-ponto é abstraída pelo servidor D-Bus, o qual implementa endereçamento e envio de mensagem para

que cada aplicação não precise determinar qual processo específico receberá a mensagem de invocação ou notificação. Esse tipo de comunicação está ilustrado na Figura 5.1

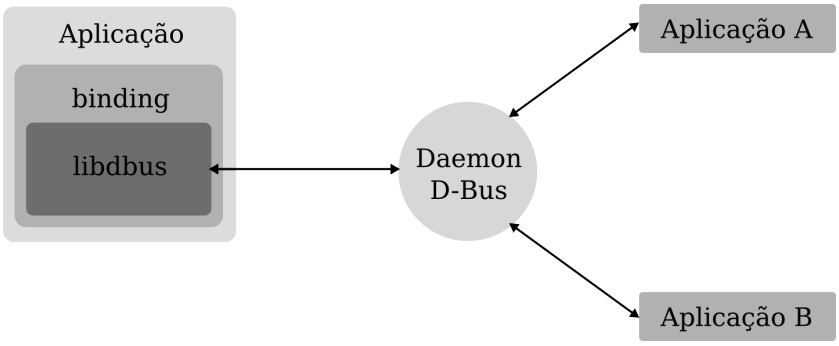

Figura 5.1: Comunicação ponto-a-ponto entre aplicações via D-Bus.

Também é possível o envio de sinais através dos barramentos. O daemon D-Bus é responsável por distribuir o sinal para as aplicações ouvintes. Tal cenário está ilustrado na Figura 5.2.

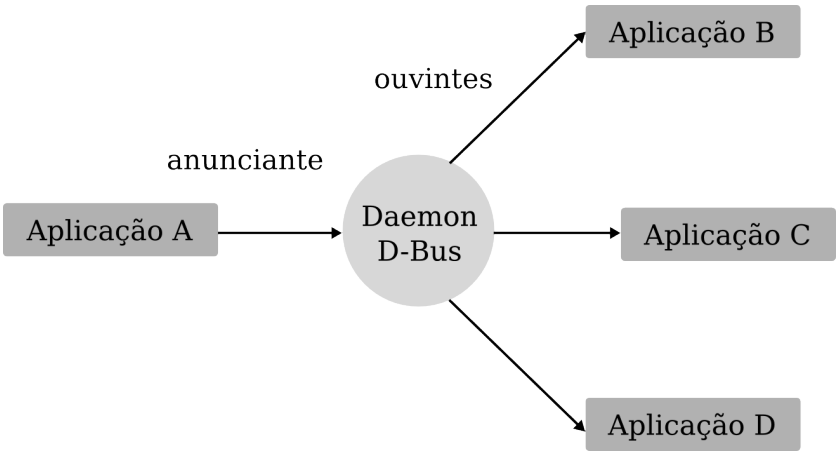

Figura 5.2: Envio de sinais por uma aplicação via D-Bus.

O envio de uma mensagem usando o D-Bus sempre envolve os seguintes passos (em condições normais de execução):

- Criação e envio da mensagem através do servidor de barramento. Tal fato deve gerar pelo menos duas mudanças de contexto;
- Processamento da mensagem pelo servidor de barramento. Dependendo do tipo da mensagem, é possível solicitar dados para reconhecimento, respondê-la ou então

ignorá-la. O último caso só é possível para notificações, isto é, sinais na terminologia D-Bus. Reconhecimento ou respostas devem gerar mais mudanças de contextos.

Assim, caso o programador pretenda transferir grandes quantidades de dados entre processos, o D-Bus não é a solução mais indicada. A melhor abordagem é a utilização de memória compartilhada, porém é geralmente mais complexa de ser devidamente implementada.

Endereçamento e Nomes no D-Bus

Para que as mensagens sejam devidamente entregues ao destinatário, o mecanismo de IPC deve suportar algumas formas de endereçamento. No D-Bus, o esquema de endereçamento é bastante flexível e eficiente. Cada barramento possui seu próprio espaço de nomes o qual não é diretamente relacionado com outros barramentos.

Para enviar uma mensagem, é necessário um endereço destino o qual é formado por uma série de outros elementos:

- O barramento por onde a mensagem é enviada. Normalmente, um barramento é aberto somente uma vez durante o ciclo de vida de uma aplicacão e será utilizado em algum momento para envio e recebimento de mensagens quando necessário. Dessa forma, o barramento alvo é uma parte transparente do endereço da mensagem (por exemplo, não é necessário especificá-lo separadamente toda vez que uma mensagem for enviada);

- Um nome conhecido para o serviço disponibilizado pelo provedor. Podemos compará-lo com um servidor DNS, o qual é responsável por associar nomes para se conectar a serviços ao invés de usar endereço IP específicos para serviços. A idéia por trás dos nomes D-Bus é bem similar, uma vez que o mesmo serviço pode ser implementado de maneiras diferentes por aplicações distintas. Contudo, deve-se considerar que grande parte dos serviços D-Bus existentes são únicos e que cada um deles provê seus próprios nomes, e a substituição de um serviço por outro não é comum;

  - Um nome conhecido é formado pelos caracteres [A-Za-z./_]. É necessária a existência de no mínimo dois pontos para separar os elementos distintos de um nome. Diferentemente do DNS, pontos não agregam nenhuma informação adicional

sobre hierarquia de serviços (zona de endereçamento). Portanto, o sistema de nomes D-Bus não é hierárquico;

  - Para diminuir confrontos no espaço de nomes D-Bus, recomenda-se que os nomes sejam formados colocando de maneira reversa os rótulos de um domínio DNS do projeto. Essa abordagem é similar à nomenclatura adotada por pacotes Java;
  - Exemplos: org.embedded.brisa e org.freedesktop.Notifications.

- Cada serviço pode conter vários objetos diferentes, e cada um pode prover serviços já existentes ou novos. Para separar um objeto dos demais, utiliza-se caminhos de objetos (*object paths*). Por exemplo, uma aplicação para gerência de informação pessoal (*Personal Information Management* - PIM) possui objetos distintos para gerenciamento das informações sobre contatos e sincronização dos mesmos;

  - Caminho de objetos parecem com caminhos de arquivos (os elementos são separadados com o caracter "/");
  - É comum a formação de caminhos de objetos usando os mesmos elementos do nome conhecido, porém os pontos (".") são substituídos pela barra ("/") e adicionando o nome específico para o objeto ao final. Por exemplo, /org/maemo/Alert/Alerter. Embora seja somente uma convenção, o esquema de nomes adotado também resolve um problema específico quando um processo é reutilizado por uma conexão D-Bus já existente sem o conhecimento explícito, por exemplo, usando uma biblioteca que encapsule as funcionalidades D-Bus. A utilização e nomes curtos poderia aumentar o risco de colisões no espaço de nomes de um processo;
  - Assim como nomes conhecidos, caminhos de objetos não possuem hierarquia, mesmo que o separador de caminhos seja utilizado. A única situação onde é possível determinar algum tipo de hierarquia, por conta dos elementos dos caminhos dos componentes, é através de uma interface instrospectiva (*introspection interface*).

- O D-Bus implementa uma unidade chamada interface no sistema de nomes. Esta

unidade é utilizada para suportar um sistema de mapeamento no qual objetos são unidades provedoras de serviços. A interface especifica os métodos que podem ser invocados, os parâmetros (chamados de argumentos no D-Bus) e possíveis sinais. É possível reusar a mesma interface para vários objetos distintos que implementam o mesmo serviço, ou geralmente, um simples objeto que implementa diferentes serviços.

  - Nomes de membros podem conter letras, dígitos e sublinhado. Por exemplo, RetrieveQuote;

- Os nomes de interfaces usam as mesmas regras de formação dos nomes conhecidos. Talvez pareça confuso no início, uma vez que nomes conhecidos são utilizados para propósitos bem distintos. Contudo, com o tempo, o programador se adequa a essas diferenças;

- Para serviços simples, é comum repetir o mesmo nome conhecido na interface. Tal cenário é bastante comum em serviços existentes;

- A última parte do endereço da mensagem é o nome do membro (*member name*), o qual também é chamado de nome do método caso um procedimento remoto seja invocado, ou então de nome do sinal, se for utilizado para emitir uma notificação. O nome do membro determina qual o procedimento a ser invocado ou o sinal a ser emitido. É necessário que o nome do membro seja único em uma interface que um objeto implementa.

  - Nomes de membros podem conter letras, dígitos e sublinhado. Por exemplo, RetrieveQuote;

- Para maiores detalhes sobre os pontos acima abordados, veja a documentação D-Bus [1].

Os tópicos explorados acima mostram as regras mais importantes para as regras de endereçamento D-Bus normalmente encontradas. A seguir, é descrito um exemplo de todos os quatro componentes descritos acima (respectivamente, nome do objeto, localização do objeto, interface e o nome conhecido), que podem ser utilizados para enviar uma simples mensagem (uma invocação de método) no SDK:

[1] D-Bus. `http://dbus.freedesktop.org/`

```
1 define SYSNOTE_NAME  "org.freedesktop.Notifications"
2 define SYSNOTE_OPATH "/org/freedesktop/Notifications"
3 define SYSNOTE_IFACE "org.freedesktop.Notifications"
4 define SYSNOTE_NOTE  "SystemNoteDialog"
```

Mesmo com o uso das funções RPC da biblioteca LibOSSO, o qual incapsula grande parte das funcionalidades D-Bus, ainda é necessário utilizar todos os componentes de nomes do framework.

5.2 D-Bus na Plataforma maemo

O D-Bus é o mecanismo de IPC *de facto* da plataforma maemo, utilizado para transferir mensagens entre os vários componentes de software. Algumas funcionalidades importantes, como gerenciamento de memória e captura de eventos oriundos da plataforma, dependem do D-Bus para o perfeito funcionamento. Além disso, o D-Bus também possibilita a utilização de serviços já existentes na plataforma maemo, tais como conexão a pontos de acesso e Bluetooth. As principais funcionalidades do D-Bus na plataforma maemo são exploradas detalhadamente nas sessões seguintes.

As funções do sistema D-Bus podem ser acessadas através da biblioteca `libdbus`. Porém, normalmente, utilizam-se as funções implementadas na biblioteca LibOSSO. Para iniciar a utilização do D-Bus, é necessário criar um contexto LibOSSO, o qual é uma estrutura que armazena informações necessárias para que as funções LibOSSO se comuniquem através do D-Bus (através de barramentos de mensagens e de sistema). Quando um contexto é criado, devemos informar o nome da aplicação. Esse nome é utilizado para registrar a aplicação que, posteriormente, pode ser encerrada para economia de memória (supondo que a aplicação tenha sido carregada a partir do Navegador de Tarefas). O nome da aplicação deve conter o caracter “.”, caso contrário, o prefixo “com.nokia” será adicionado ao nome da aplicação. É importante que não haja colisão de nomes de aplicações, por isso, recomenda-se que o nome seja baseado em um domínio DNS de seu próprio controle. Se você for implementar serviços D-Bus para uso por clientes (com as funções `osso_rpc_set_cb`), é necessário cuidado extra ao escolher o nome da aplicação. No código a seguir, um contexto LibOSSO para uso posterior (em C e Python, respectivamente) é inicializado.

```c
#include <libosso.h>
...
int main(int argc, char *argv[]) {
    osso_context_t *osso_context;
    osso_return_t result;
...
    osso_context = osso_initialize("MinhaAplicacao", "0.0.1", TRUE, NULL)
        ;
    if (osso_context == NULL) {
        return OSSO_ERROR;
    }
...
}
```

```python
import osso
...
def main():
  osso_c = osso.Context("MinhaAplicacao", "0.1", False)
  ..
```

5.2.1 Mensagens de Estado do Hardware

As notificações de mudança de estado do hardware (bateria, cartão de memória removido/inserido, cabo USB conectado/desconectado, etc.) são enviadas através do sistema D-Bus. Dessa forma, o programador é capaz de capturar o sinal referente a bateria fraca e tomar alguma decisão, por exemplo, salvar o arquivo que está sendo atualmente editado. No código abaixo (C e Python, respectivamente), os sinais do estado do hardware são capturados.

```c
...

void hw_event_handler(osso_hw_state_t *state, gpointer data) {
    GtkWidget *window;
    window = (GtkWidget*) data;

    if (state->shutdown_ind) {
```

```
8           hildon_banner_show_information(GTK_WIDGET(window), NULL, "
                Desligar");
9       } if (state->memory_low_ind) {
10          hildon_banner_show_information(GTK_WIDGET(window), NULL, "Baixa
                memoria");
11      } if (state->save_unsaved_data_ind) {
12          hildon_banner_show_information(GTK_WIDGET(window), NULL, "Salvar
                dados");
13      } if (state->system_inactivity_ind) {
14          hildon_banner_show_information(GTK_WIDGET(window), NULL, "
                Inatividade do sistema");
15      }
16  }
17  ...
18      result = osso_hw_set_event_cb(osso_context, NULL, hw_event_handler, (
            gpointer) window );
19
20      if (result != OSSO_OK) {
21          g_print("Erro ao modificar a fun\c{c}ao de chamada para eventos
                do hardware (%d)\n", result);
22          return OSSO_ERROR;
23      }
24  ...
```

```
1   ...
2
3   def state_cb(shutdown, save_unsaved_data, memory_low, system_inactivity,
4       message, loop):
5       print "Desligar: ", shutdown
6       print "Salvar dados: ", save_unsaved_data
7       print "Baixa memoria: ", memory_low
8       print "Inatividade do sistema: ", system_inactivity
9       print "Messagem: ", message
10  ...
11  def main():
12  ...
13      loop = gobject.MainLoop()
14      device = osso.DeviceState(osso_c)
```

```
15      device.set_device_state_callback(state_cb, user_data=loop)
16      loop.run()
17      device.set_device_state_callback(None)
```

5.2.2 Inicialização de Aplicativos

O D-Bus é responsável por carregar as aplicações através do Navegador de Tarefas e registrá-las no sistema. É necessário o registro das aplicações pelo D-Bus para tomar decisões relativas ao gerenciamento correto de memória. Maiores detalhes sobre essa funcionalidade estão descritos no Capítulo 2.

5.2.3 Finalização de Aplicativos

É possível também enviar uma mensagem de sistema para que a aplicação seja finalizada. Essa mensagem também é utilizada pelo próprio sistema. O código abaixo ilustra como isso pode ser feito (em C).

```
1  ...
2
3  void exit_event_handler(gboolean die_now, gpointer data) {
4      GtkWidget *window;
5      window = (GtkWidget *) data;
6      g_print("invocando exit_event_handler\n");
7      hildon_banner_show_information(GTK_WIDGET(window), NULL, "Finalizando
           ...");
8  }
9
10 ...
11
12 int main( int argc, char* argv[] ) {
13 ...
14     result = osso_application_set_exit_cb(osso_context,
15                                           exit_event_handler,
16                                           (gpointer) appdata);
17     if (result != OSSO_OK) {
```

```
18          g_print("Erro ao modificar a fun\c{c}ao de finaliza\c{c}ao (%d)\n
                ", result);
19          return OSSO_ERROR;
20      }
21  ...
22  }
```

5.2.4 Armazenamento do Estado da Aplicação

Trata-se de uma funcionalidade especial da plataforma maemo, anteriormente citada no Capítulo 2. É possível que as aplicações armazenem os seus estados em memória principal ou secundária (por exemplo, memória flash) e então finalizem o processo de execução. Essa atividade de armazenamento ocorre quando há trocas de aplicações ativas na área principal e a memória do dispositivo está baixa. Quando o usuário seleciona uma aplicação que estava em background, ela é inicializada novamente e o seu estado anterior é recuperado a partir dos dados salvos em memória. Dessa forma, o usuário não percebe que a aplicação estava parada por muito tempo. A biblioteca LibOSSO também implementa funções que armazenam e permitem recuperar o estado da aplicação via D-Bus.

5.3 Conclusão

O sistema D-Bus possui um papel importante na plataforma maemo, sendo responsável pela notificação de eventos da plataforma, comunicação entre as aplicações e também carregamento de aplicativos através do Navegador de Tarefas. Além disso, auxilia na reutilização de serviços bastante importantes e comuns na plataforma maemo (diálogos para conexão com pontos de acesso e dispositivos bluetooth, gerência de conexões, dentre outros), aumentando a produtividade e também a coerência dos aplicativos que executam na plataforma.

Capítulo 6

Interface Gráfica

A interface gráfica com o usuário é um ponto fundamental na concepção de produtos de alta qualidade para o usuário final. Uma aplicação bem aceita pelo usuário possui uma interface gráfica bastante funcional. Neste capítulo são discutidos os componentes gráficos da plataforma maemo, bem como o modelo no qual se baseia grande parte das bibliotecas: o modelo dirigido a eventos. Outras bibliotecas importantes recentemente portadas para a plataforma, tais como Edje/Evas e Qt, também serão apresentadas.

6.1 Introdução

Ao comparar um programa simples com interface textual e uma aplicação com interface gráfica na plataforma maemo, percebemos o uso extensivo de diferentes bibliotecas para os programas gráficos. A plataforma maemo provê muitas APIs para geração de elementos gráficos, gerência de recursos e integração do software com o framework de aplicações. Tais APIs escondem a complexidade da biblioteca X [77], com a qual programas com interface gráfica não precisam se preocupar, embora seja utilizada internamente pelas bibliotecas gráficas existentes. A decomposição de um simples programa, para a plataforma maemo, com interface gráfica é ilustrada na Figura 6.1.

A aplicação utiliza diretamente as APIs gráficas GTK+ e Hildon, além de outras bibliotecas importantes, por exemplo, o D-Bus. Perceba que a aplicação gráfica não utiliza diretamente a biblioteca X. Esse trabalho é realizado pelas bibliotecas gráficas que encapsulam as funcionalidades necessárias para o tratamento de gráficos.

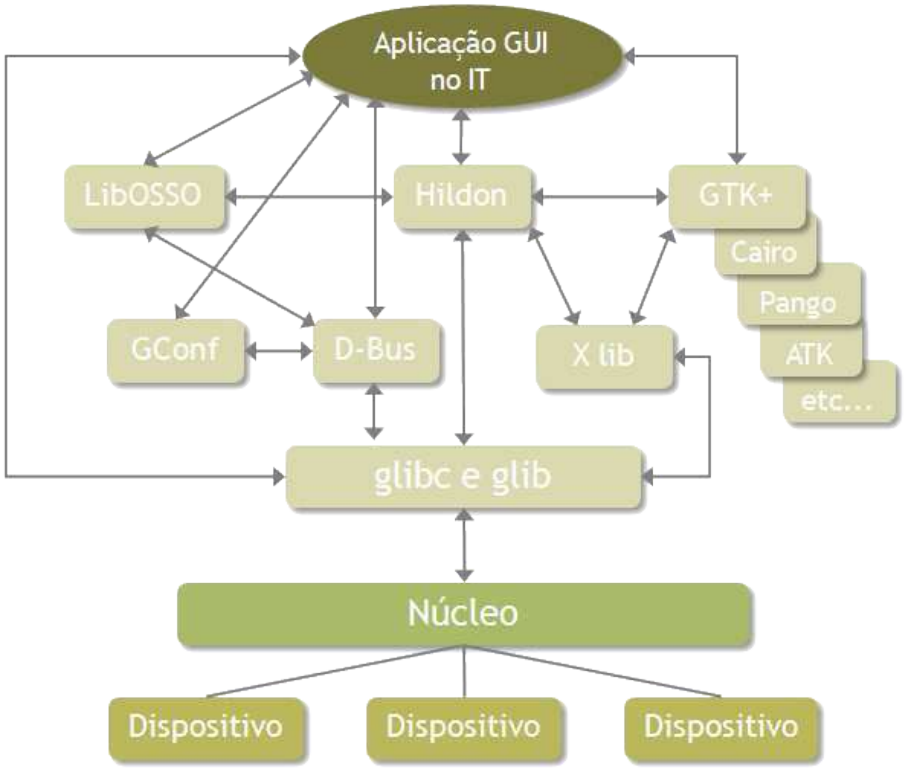

Figura 6.1: Decomposição de uma aplicação simples com interface gráfica.

6.2 Componentes gráficos da plataforma maemo

O framework de aplicações de interface gráfica da plataforma maemo é chamado Hildon. É baseado nas mesmas tecnologias com as quais o framework GNOME, bastante comum para desktops Linux, foi construído; um exemplo é a biblioteca GTK+. Hildon contém muitas melhorias com o intuito de tornar o GTK+ mais adequado para ser utilizado nos Internet Tablets: *widgets* Hildon, engine de tema Sapwood, servidor de imagens, navegador de tarefas, painel de controle, barra de status, método de entrada através de tela sensível ao toque e gerenciamento de janelas em uma tela de alta resolução. Todas essas modificações tornam as aplicações mais atraentes para o usuário final: cores mais claras e fontes maiores, por exemplo.

As APIs de programação de interface gráfica para maemo são baseadas em GTK+ e em extensões Hildon. Muitos *widgets* e métodos GTK+ funcionam no ambiente Hildon sem modificações, com algumas exceções, como por exemplo, a janela principal da aplicação a qual é substituída por uma janela Hildon.

Somente uma aplicação é visível por vez, pois a janela da aplicação ocupa toda a área de tela disponível. O Navegador de Tarefas é o responsável pelas trocas entre as aplicações e também incluem menus para carregá-las. A área da Barra de Status (barra de títulos) inclui um menu de aplicação e alguns botões para minimizar e fechar aplicativos em execução. A aplicação só possui um único menu, com submenus que se expandem horizontalmente para a direita. Dessa forma, o desenvolvedor deve planejar os menus de maneira cuidadosa, pois o espaço de menus é limitado. O teclado virtual é carregado automaticamente quando o usuário do Internet Tablet ativa uma área de entrada de textos. A janela do aplicativo em execução é adaptada para que a área para o teclado virtual também apareça na tela.

Para que a barra de status/título seja expandida através de componentes (*plug-ins*) definidos pelo usuário, é necessário prover informação do estado das aplicações. A janela principal (visível quando nenhuma aplicação estiver executando ou quando todas estiverem minimizadas) permite a execução desses componentes, também chamados de *home applets*. Na maioria das vezes, tais aplicativos fornecem informações simples, tais como notícias (leitor RSS) e informações sobre clima ou hora.

6.3 *Layouts* Gráficos

A plataforma oferece 4 *layouts* gráficos possíveis para as aplicações. As aplicações podem passar para o modo tela cheia a qualquer momento, e também voltar para o estado anterior. No *layout* padrão, a aplicação tem uma área útil de 696x396 pixels e as áreas do Navegador de Tarefas e Barra de Status/Título são visíveis, conforme ilustrado na Figura 6.2.

No *layout* ilustrado na Figura 6.3, uma barra de ferramentas é inserida e as áreas do Navegador de Tarefas e da Barra de Status/Título continuam visíveis. Porém, a área da aplicação diminui, chegando a 696x360 pixels.

Os layouts de tela cheia podem ser ativados através de eventos de hardware (botões) ou de software implementados pelo programador. Em ambos, a área da aplicação é bem maior, chegando até 800x480 pixels. Contudo, as áreas do Navegador de Tarefas e da Barra de Status/Título ficam escondidas. Os layouts de tela cheia são ilustrados nas Figuras 6.4 e 6.5.

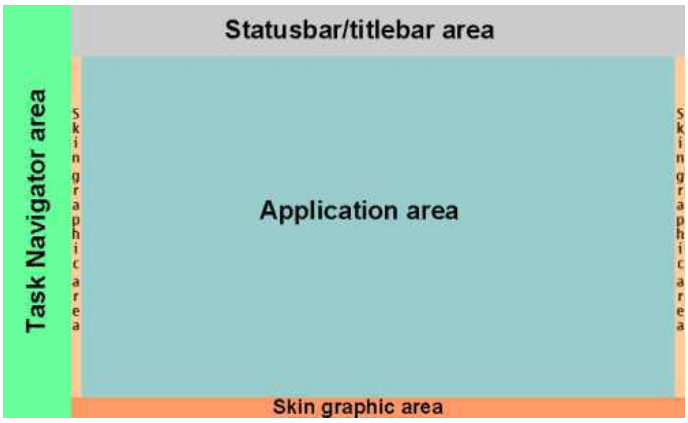

Figura 6.2: Layout padrão.

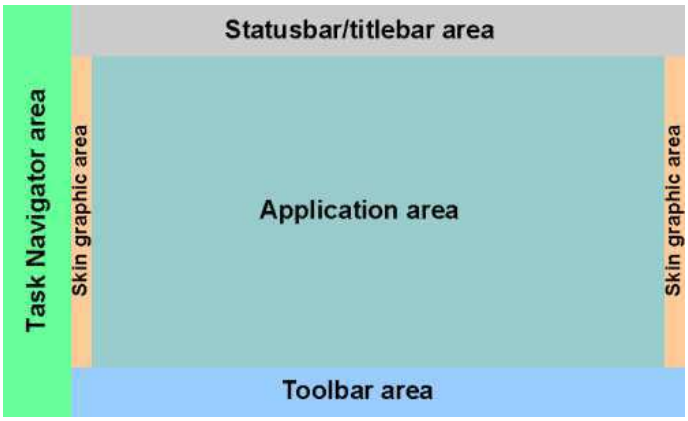

Figura 6.3: Layout padrão com barra de ferramentas.

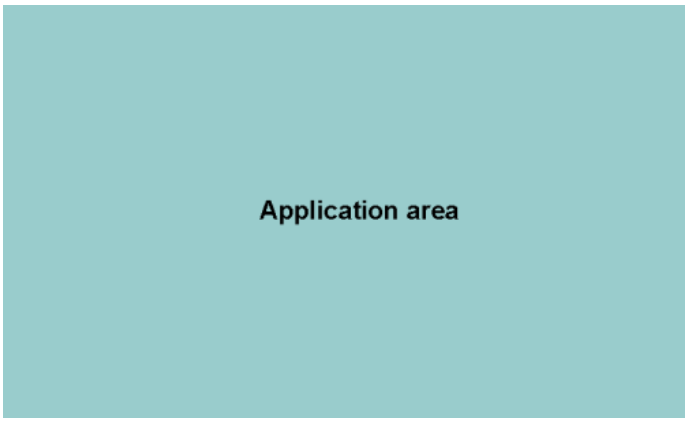

Figura 6.4: Layout padrão em modo tela cheia.

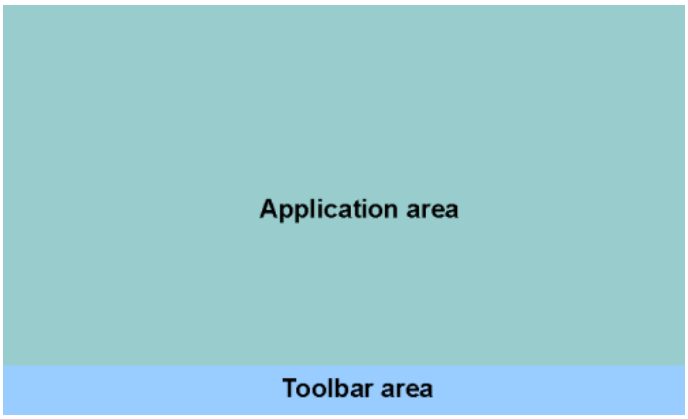

Figura 6.5: Layout padrão em modo tela cheia com barra de ferramentas.

6.3.1 Modelo baseado em eventos

Para permitir a interação entre os vários componentes (por exemplo, gráficos ou do sistema), diferentes bibliotecas implementam maneiras distintas para notificação de eventos e de mudanças. Cada uma tem diferenças de *design* e algumas restrições. O modelo usado pela biblioteca GTK+ é baseado em uma abordagem de registro/auto-entrega com chamadas a funções implementadas antecipadamente que realizam o tratamento das notificações.

Este mecanismo é chamado de GSignal e é implementado pela biblioteca GObject. O GObject é um framework e uma biblioteca de infra-estrutura de aplicações para a linguagem C que é usado para implementar software orientado a objetos de maneira similar aos desenvolvidos com Java e C++. Isto é realizado de maneira portável sem a necessidade de passos anteriores para processamento dos arquivos fontes, diferentemente de outras soluções mais complexas usadas em outras bibliotecas. Uma vez que nenhuma ferramenta especial é necessária, GObject é uma solução bastante portável. Também utiliza a biblioteca GLib exclusivamente para implementação das estrutras de dados e gerência de memória.

Questões relativas ao uso mais avançado da biblioteca GObject (implementação dos próprios tipos, extensão de classes, etc.) não são cobertas nesse material. O leitor pode obter maiores referências em [88; 47; 72].

Widgets GTK+ (e também *widgets* Hildon) usam o mecanismo de sinal GSignal para implementar mensagens de notificação que são geradas por *widgets* quando algo ocorre (normalmente uma ação gerada por um evento de interação do usuário com a interface gráfica).

Essas mensagens são enviadas para os ouvintes interessados através de funções de chamadas (*callback functions*). Não há um barramento único para compartilhamento de eventos, mas um sistema no qual você pode adicionar ligações entre os *widgets* e as funções de chamada desejadas. GSignal também oferece um mecanismo bastante eficiente para especificar em qual momento da entrega de sinais o sistema deve invocar uma chamada específica, sendo possível a definição de prioridades.

Cada classe define quais os tipos de sinais que uma instância da classe pode emitir (enviar). Os sinais são identificados por uma string e a fonte do sinal, que pode ser tanto um objeto que tenha algo interessante para notificar, ou um componente importante do sistema. O programador "conecta um sinal" especificando o objeto que é capaz de emitir um sinal. Também é necessário dar um nome ao sinal. O nome é utilizado somente para diferenciar entre os diversos tipos de sinais, como "clicked" ou "selected". Na Figura 6.6 são ilustrados os sinais e respectivas funções de chamada do widget GtkButton.

```
"activate"  void        user_function      (GtkButton *widget,
                                            gpointer   user_data)
"clicked"   void        user_function      (GtkButton *button,
                                            gpointer   user_data)
"enter"     void        user_function      (GtkButton *button,
                                            gpointer   user_data)
"leave"     void        user_function      (GtkButton *button,
                                            gpointer   user_data)
"pressed"   void        user_function      (GtkButton *button,
                                            gpointer   user_data)
"released"  void        user_function      (GtkButton *button,
                                            gpointer   user_data)
```

Figura 6.6: Sinais e respectivas funções de chamada para o widget GtkButton.

O mecanismo provido pelo GSignal está ilustrado na Figura 6.7. Perceba que a aplicação fica em estado de espera (laço principal), até que um evento seja gerado por alguma widget da interface gráfica. Uma vez que o sinal está conectado, é necessário um alvo para essa conexão. Esse alvo é a função que implementa o código que deverá ser invocado toda vez que um determinado sinal for emitido. O objeto que emite o sinal não invoca a função de chamada diretamente; para isso, o objeto utiliza um framework genérico de entrega de sinal provido pelo GObject. Assim, obtem-se interfaces mais simples tanto para o emissor quanto para o receptor, como também para conexão de sinais. O código abaixo (em C e logo depois em Python) ilustra como deve ser feito para associar uma função de tratamento e um sinal

enviado por um GtkButton sempre que ele é clicado.

```
#include <stdlib.h>
#include <gtk/gtk.h>

void button_callback(GtkWidget* source, gpointer data) {
    g_printf("O botao foi clicado!\n");
}

int main(int argc, char** argv) {
...
    GtkWidget* button = gtk_button_new_with_label("Botao")
    g_signal_connect(G_OBJECT(button), "clicked", G_CALLBACK(
        button_callback), NULL);
...
}
```

```
import gtk

...
def button_callback(*args):
    print 'O botao foi clicado!\n'
...
    button = gtk.Button("Botao")
    button.connect('clicked',button_callback)
...
```

Sempre que o botão for clicado, ele gera um evento, cujo nome do sinal é “clicked”. Dessa forma, quando a aplicação recebe a notificação do evento, ela executa o código da função `button_callback`.

Dado que a linguagem C foi criada antes que a programação orientada a objeto tenha amadurecido, GObject e sinais são uma solução bastante elegante. Entretanto, se o programador tiver experiência apenas com linguagens orientadas a objeto, GObject parece ser uma solução estranha. Porém, os conceitos são os mesmos e o que muda é apenas a sintaxe.

Sinais não é algo inerente à biblioteca GTK+. Na verdade, a própria documentação do GObject define sinal como “um meio de customização do comportamento de um objeto e um mecanismo de propósito geral para notificação”. Para um estudo mais detalhado sobre

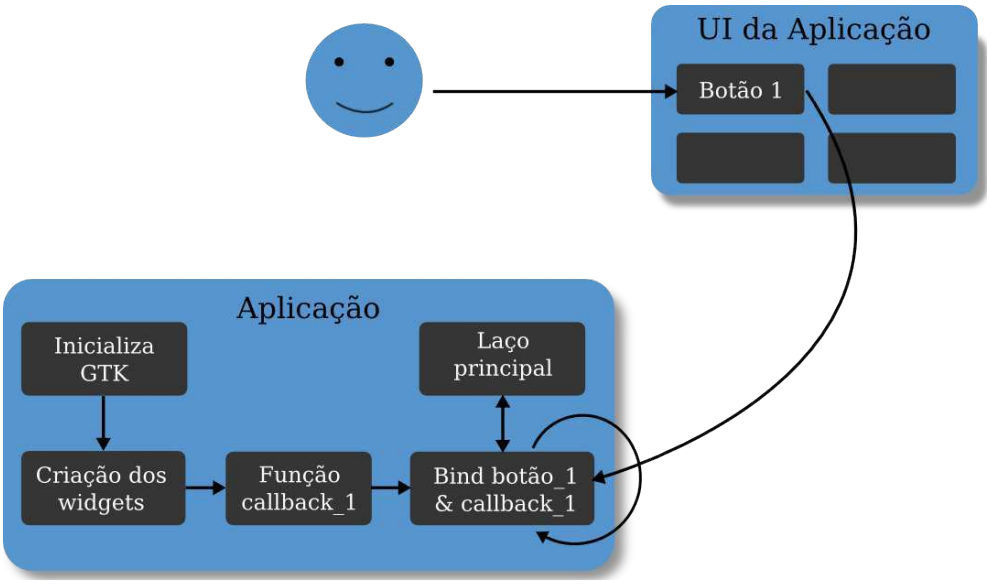

Figura 6.7: Notificação de um evento gerado pela interação do usuário com a interface gráfica.

a biblioteca GObject, o programdor pode consultar a documentação da API ou alguns dos vários livros que explicam a biblioteca detalhadamente [88].

6.3.2 Módulos Principais

GLib

Todos os programas GTK+ e Hildon usam a biblioteca utilitária GLib, a qual provê um conjunto de tipos portáveis para programas escritos na linguagem C, bem como uma série de funções utilitárias. Assim, a biblioteca facilita o desenvolvimento de software mais portável de maneira mais fácil, reduzindo a necessidade de reescrever código que realiza a mesma função mas para plataformas distintas. Na Tabela 6.1 estão listados os principais tipos da biblioteca.

Além dos tipos, a biblioteca GLib também oferece algumas funcionalidades bastante úteis, por exemplo: alocação de memória; saída de mensagens, depuração e funções para logging; biblioteca de threads; operações de strings; operações de data e tempo e estrutura de dados (listas encadeadas, tabelas hash, arrays dinâmicos, etc.).

A documentação completa da biblioteca gLib pode ser acessada através do endereço

Tabela 6.1: Tipos de dados da biblioteca GLib.

Tipo	Descrição
gboolean	Pode assumir o valor TRUE ou FALSE
gint8	Inteiro de 8 bits com sinal
gint16	Inteiro de 16 bits com sinal
gint32	Inteiro de 32 bits com sinal
gint64	Inteiro de 64 bits com sinal
gpointer	Ponteiro sem tipo (void *)
gconstpointer	Ponteiro sem tipo somente para leitura (const void *)
gchar	Tipo 'char' do compilador (8 bits no gcc)
guchar	Tipo 'unsigned char'
gshort	Tipo 'short' do compilador (16 bits no gcc)
gushort	Tipo 'unsigned short'
gint	Tipo 'int' do compilador (normalmente 32 bits no gcc)
guint	Tipo 'unsigned int'
glong	Tipo 'long' do compilador (32/64 bits no gcc)
gulong	Tipo 'unsigned long'
gfloat	Tipo 'float' do compilador (32 bits no gcc)
gdouble	Tipo 'double' do compilador (64/80/81 bits no gcc)

`http://maemo.org/api_refs/4.0/glib/index.html`. Para outras bibliotecas da plataforma maemo, acesse a página principal de APIs em `http://maemo.org/development/documentation/apis/4-x/`.

GTK+

O GTK+ é uma biblioteca para criação de interfaces gráficas para o usuário, escrita em C e com uma arquitetura orientada a objetos. Portada para várias plataformas como Linux e Windows, a biblioteca GTK+ é liberada sob licença LGPL (*GNU Library General Public License*). Dessa forma, as aplicações que utilizam o GTK+ podem ter licenças mais restritivas para uso comercial. Há bindings para várias outras linguagens, incluindo C++, Perl, Python, Ada95, Pascal e Eiffel. O GTK+ depende dos seguintes componentes:

GLib Biblioteca utilitária descrita na subseção anterior;

Pango É uma biblioteca para localização de texto. Toda a funcionalidade é centrada no objeto PangoLayout, o qual representa um parágrafo de texto. Pango provê as funções básicas para GtkTextView, GtkLabel, GtkEntry, e outros widgets que disponibilizam texto;

ATK Framework de acessibilidade. Provê um conjunto de interfaces genéricas para uso de tecnologias de acessibilidade que permitem a interação com a interface gráfica. Por exemplo, um leitor de tela usa o ATK para descobrir qual texto está escrito e reproduzi-lo para usuários cegos.

GdkPixbuf Biblioteca de tamanho pequeno que permite a criação de objetos GdkPixbuf ("pixel buffer") a partir de dados ou arquivo de imagens. Para disponibilizar imagens, utiliza-se GdkPixbuf em combinação com um GtkImage;

GDK Camada abstrata a qual permite que a biblioteca GTK+ suporte vários sistemas de janelamento. GDK provê funcionalidades para desenhar gráficos e outras facilidades para os sistemas X11, Windows e outros dispositivos Linux.

GTK+ permite a criação de aplicações gráficas de maneira simples. O código abaixo cria uma janela (GtkWindow) e coloca um GtkLabel no centro. A aplicação resultante é ilustrada na Figura 6.8.

```
1  #include <gtk/gtk.h>
2
3  int main(int argc, char *argv[]) {
4      GtkWidget *window;
5
6      gtk_init(&argc, &argv);
7
8      window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
9      gtk_window_set_title(GTK_WINDOW(window),"Hello GTK+")
10     g_signal_connect (G_OBJECT (window), "destroy",
11                       G_CALLBACK(gtk_main_quit), NULL);
12
13     gtk_container_set_border_width(GTK_CONTAINER (window), 10);
14     gtk_container_add(GTK_CONTAINER(window),
15                       gtk_label_new("Hello World"));
16
17     gtk_widget_show_all(window);
18     gtk_main();
19
20     return 0;
21 }
```

A aplicação executa normalmente. Entretanto, não parece com uma aplicação com o tema Hildon. A tela de fundo é escura e as fontes são pequenas, por exemplo. É necessário executar a aplicação usando um *script* utilitário que modifica para o tema correto: `run-standalone.sh`. Na Figura 6.9, a aplicação anterior está sendo executada com o tema Hildon.

Hildon

A plataforma também contém um conjunto de *widgets* que foram otimizados para dispositivos com interface gráfica limitada e interação com o usuário distinta (tela sensível ao toque e auxílio de uma caneta). A tela é fisicamente pequena se comparada às telas modernas para PCs. Dessa forma, são necessárias soluções diferentes para que seja possível intercalar o uso das diversas aplicações com interface gráfica e definir quando um gerenciador de tela ou o

Figura 6.8: Aplicação simples em GTK+ sendo executada no ambiente desktop.

Figura 6.9: Aplicação simples em GTK+ sendo executada no ambiente desktop com o tema Hildon.

desktop gráfico estão disponíveis. Para integrar a aplicação corretamente ao ambiente AF (*application framework*), devemos modificar o modelo do software e adicionar os widgets Hildon.

O Hildon provê dois widgets principais: HildonProgram e HildonWindows, os quais substituem algumas funcionalidades providas pelo widget GtkWindow. O HildonProgram é

uma "super-janela" que provê um shell pelo qual é possível integrar as visões gráficas com o ambiente de execução. Visões são similares a janelas, mas somente uma visão é visível ao mesmo tempo. Podemos comparar uma aplicação Hildon com um diálogo com várias *tabs*, uma para cada janela. Porém, neste caso, as *tabs* são a única interface visível pelo usuário. Estas visões são implementadas utilizando widgets HildonWindow, os quais são basicamente contêineres nos quais se adiciona outros widgets (botões, caixas de texto, etc.). Dessa forma, caso alguma aplicação gráfica existente no desktop seja portada para a plataforma maemo, é necessário que alguns widgets sejam substituídos por elementos Hildon. Há outros pontos importantes que devem ser considerados:

- Cada visão pode ter apenas um GtkMenu que a aplicação possa utilizar, ou seja, há apenas uma lista de itens de menu possível;
- Cada janela possui um contêiner no qual podem ser adicionados outros elementos de interface gráfica. Entretanto, não é possível adicionar mais de um elemento na janela, pois apenas um será visualizado. Para resolver essa limitação, utiliza-se objetos da classe GtkBox (GtkVBox, GtkHBox), os quais permitem a adição de vários elementos de interface;
- Quando algum campo de entrada de texto ganha foco, o teclado virtual é ativado automaticamente. Com isso, a área do layout da aplicação é redimensionado. Diálogos abertos pela aplicação podem também ser modificados, caso necessário. Porém, caso o programador não deseje que o layout da aplicação seja redimensionado por causa do teclado virtual, é possível inseri-la dentro de uma GtkScrolledWindow [1];
- Evitar muitos submenus dentro do menu principal, pois a área da janela é limitada;
- Criar a aplicação para que apenas uma janela principal esteja visível por vez. Aplicativos que possuem múltiplas janelas ou diálogos distintos, que abrem simutaneamente, devem ser projetados novamente;
- Evitar disponibilizar muita informação ao mesmo tempo.

[1] http://maemo.org/api_refs/4.0/gtk/GtkScrolledWindow.html

É possível modificar o exemplo anterior "Hello World" para utilizar os widgets HildonProgram e HildonWindow. O resultado está descrito no código a seguir:

```
1  #include <hildon/hildon-program.h>
2  #include <gtk/gtkmain.h>
3  #include <gtk/gtkbutton.h>
4
5  int main(int argc, char *argv[]) {
6      HildonProgram *program;
7      HildonWindow *window;
8      GtkWidget *button;
9
10     gtk_init(&argc, &argv);
11
12     program = HILDON_PROGRAM(hildon_program_get_instance());
13     g_set_application_name("Hello World!");
14
15     window = HILDON_WINDOW(hildon_window_new());
16     hildon_program_add_window(program, window);
17
18     button = gtk_button_new_with_label("Hello!");
19     gtk_container_add(GTK_CONTAINER(window), button);
20
21     g_signal_connect(G_OBJECT(window), "delete_event",
22                      G_CALLBACK(gtk_main_quit), NULL);
23
24     gtk_widget_show_all(GTK_WIDGET(window));
25     gtk_main();
26
27     return 0;
28 }
```

Perceba que foi utilizada um HildonWindow no lugar do GtkWindow. A utilização do HildonWindow é bastante semelhante ao uso do GtkWindow: modificação de propriedades, adição de widgets, etc.

6.4 Outras APIs de Interface Gráfica

A pilha de bibliotecas da plataforma maemo não está limitada àquelas apresentadas no Capítulo 2. É possível a adição de componentes externos já presentes em ambientes desktop. A seguir, são descritas algumas bibliotecas de interface gráfica comumente usadas na plataforma maemo como opções para GTK+ e Hildon. Embora as bibliotecas padrão sejam bastante úteis e possuam funcionalidades interesantes, não possibilitam o desenvolvimento de interfaces gráficas mais atrativas para o usuário.

6.4.1 SDL

A biblioteca foi criada por Sam Lantinga e a primeira release foi lançada no início de 1998, enquanto empregado da Loki Software. A idéia surgiu ao portar uma aplicação Windows para Mac. Ele utilizou o SDL para portar o jogo Doom para o BeOS. Pouco tempo depois, algumas bibliotecas passaram a utilizar o SDL, como o SMPEG e OpenAL. A biblioteca é bastante utilizada para o desenvolvimento de jogos e possui uma série de funções que auxiliam o trabalho dos desenvolvedores de jogos: por exemplo, temporizador e acesso a funções de multimídia.

A biblioteca SDL possui vínculos para outras linguagens de programação: C++, Perl, Python, Euphoria, Pliant, dentre outras. Devido a essa característica e ao fato de ser um projeto *open source*, licenciado sob LGPL, o SDL torna-se uma escolha bastante comum para muitas aplicações multimídias. A biblioteca comporta-se, basicamente, como uma camada simples que provê suporte para operações em gráficos, sons, acesso a arquivos, captura de eventos, temporizador, gerência de threads e outras funcionalidades. Comumente, a biblioteca é utilizada juntamente com o OpenGL para manipular a saída gráfica e permitir a leitura dos dados do teclado e mouse, os quais estão fora do scopo do OpenGL.

A biblioteca é dividida em alguns subsistemas: vídeo (implementa funções para manipulação de superfície e tratamento com o framework OpenGL), áudio, cd-rom, joystick e temporizador. Além do suporte básico e de baixo nível, há também algumas bibliotecas oficiais distintas que provêem funcionalidades adicionais. Estas bibliotecas fazem parte da biblioteca padrão e estão disponibilizadas no site oficial:

- SDL_image - módulo de suporte para vários formatos de imagem;

Figura 6.10: Versão inicial do aplicativo Canola desenvolvido com SDL.

- SDL_mixer - funções para manipulação de aúdio, principalmente mixagem de som;
- SDL_net - módulo de suporte à rede;
- SDL_ttf - renderização de fontes TrueType
- SDL_rtf - renderização de fontes Rich Text

6.4.2 Edje/Evas

O projeto *Enlightenment Foundation Libraries* (EFL) é um conjunto de bibliotecas gráficas *open source* baseado no gerenciador de janelas Enlightenment. Foi desenvolvido pelo Enlightenment.org com patrocínio da empresa Terra Soft Solutions. O principal objetivo do projeto é tornar o EFL uma solução flexível e fácil de ser utilizada. As bibliotecas foram criadas a partir da versão 0.17. de gerenciador de janelas. As bibliotecas foram criadas para serem portáveis e otimizadas, podendo ser utilizadas em qualquer plataforma, até mesmo as embarcadas, como os Internet Tablets.

Evas é a biblioteca canvas do EFL para a criação de áreas ou janelas em um Sistema de Janelas X. O EFL usa o módulo de aceleração gráfica da plataforma sempre que possível para obter melhores resultados. Porém, também se adequa a hardware mais limitado, diminuindo a variedade de cores e a qualidade dos gráficos se necessário. Diferentemente da maioria

Figura 6.11: Versão mais atual do aplicativo Canola desenvolvido com a biblioteca Evas/Edje.

das bibliotecas canvas, o Evas é baseado em imagem (ao contrário das bibliotecas baseadas em vetores) e consciente do estado dos elementos de interface (a grande maioria das bibliotecas canvas não são sensíveis à mudança do estado das aplicações, sendo o programador responsável por esta tarefa).

Edje é uma biblioteca que procura separar a interface gráfica da lógica da aplicação. Dessa forma, é possível modificar a interface gráfica sem alterar o código que implementa a lógica da aplicação. Permite que as aplicações possuam temas, ou seja, botões, janelas e outros elementos de interface gráfica do software possuem o mesmo padrão (cores, fontes, formas, etc.). Ao utilizarmos a biblioteca Edje em um determinado projeto, as especificações do *layout* de interface são mantidas em um arquivo (separado do código fonte da aplicação).

6.4.3 Qt

Desenvolvido pela empresa norueguesa Trolltech, Qt (pronuncia-se *cute*) [14] é um framework de desenvolvimento de aplicações, bastante utilizado para a criação de programas com interface gráfica e também aplicações sem interface gráfica, como ferramentas de console e servidores. Qt é utilizado pelo KDE, pelo navegador web Opera, Google Earth, Skype, Qtopia, Photoshop Elements e OPIE. Recentemente, a Nokia adquiriu a Trolltech, e futu-

ramente deve incorporar o Qt em suas plataformas (Symbian, maemo, etc.). Com isso, em pouco tempo, estará disponível uma outra opção em biblioteca gráfica para o desenvolvimento de aplicações para a plataforma maemo.

Qt usa a linguagem C++ com várias extensões implementadas por um pré-processador que gera um código C++ padrão após a compilação. É possível desenvolver com Qt também em outras linguagens de programação. Existem vínculos para Ada, C#, Java, Pascal, Perl, PHP, Ruby e Python. Qt é multi-plataforma e possui um suporte para localização. Como citado anteriormente, há também funcionalidades que não são específicas de interface gráfica: acesso a banco de dados SQL, parser XML, gerência de threads, dentre outros.

6.5 Exemplo de aplicações

Grande parte das aplicações maemo são desenvolvidas usando os frameworks GTK+/Hildon. Essa preferência decorre da boa documentação existente, bem como de vários exemplos já desenvolvidos. Alguns desses exemplos de aplicações estão ilustrados na Figura 6.12. Embora o framework GTK+/Hildon possua vários recursos que permitem a criação de diversas interfaces gráficas, algumas aplicações exigem um layout gráfico mais complexo e dinâmico, os quais podem ser oferecidos pela biblioteca Evas/Edje, por exemplo. Conforme discutido anteriormente, Evas/Edje e SDL possibilitam o desenvolvimento de aplicações com uma interface gráfica bem mais complexa e atraente para o usuário final. Contudo, essas bibliotecas são pouco usadas devido a carência de documentação (Evas/Edje) ou mesmo pela necessidade de um maior tempo de aprendizagem para utilizá-las. As Figuras 6.10 e 6.11 ilustram exemplos de uma aplicação desenvolvida em SDL e Evas/Edje, respectivamente.

6.6 Conclusão

Atualmente, interface gráfica de aplicações, especialmente para a plataforma maemo, estão sendo tratadas com maior atenção. Percebeu-se que aplicações com layout gráfico atrativo possuem maior aceitação entre os usuários, mesmo que as problemas com funcionalidades e bugs estejam presentes. Embora a plataforma maemo disponibilize o framework GTK+/Hildon, o qual não permite a criação de intefaces gráficas mais sofisticadas, é possível

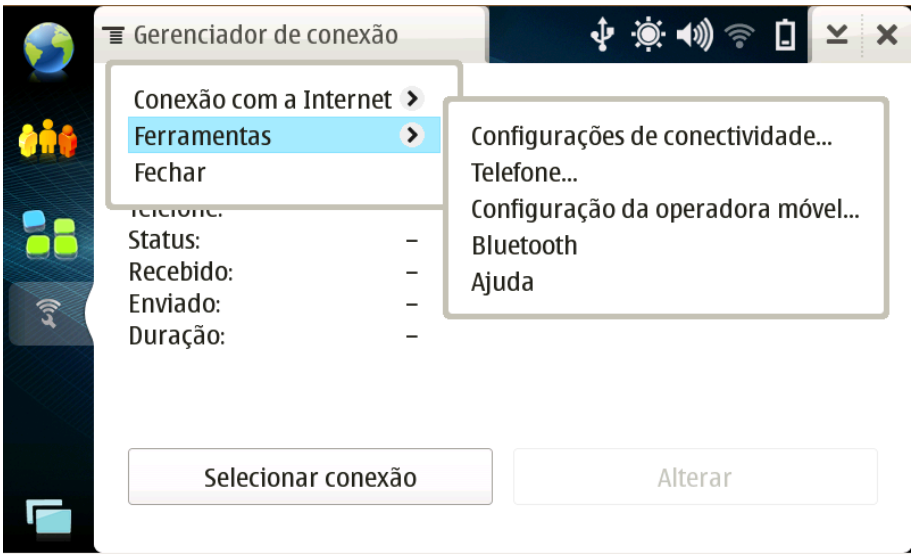

Figura 6.12: Exemplo de aplicação desenvolvida com Gtk+/Hildon.

criar aplicação com o layout gráfico bastante intuitivo. Dessa forma, o usuário sente-se mais familiarizado com a aplicação. Caso seja necessário uma interface gráfica mais complexa (com movimento e menus flutuantes, por exemplo), a plataforma maemo também disponibiliza soluçoẽs como as bibliotecas SDL e Evas/Edje.

Capítulo 7

Multimídia

Os Internet Tablets possuem uma arquitetura completa o bastante para oferecer serviços de multimídia de qualidade, tais como execução de vários formatos de áudio e vídeo. A plataforma maemo oferece uma série de bibliotecas e módulos que auxiliam o desenvolvimento de aplicações multimídia de maneira fácil e rápida. Neste capítulo serão apresentados detalhes do subsistema de multimidia da plataforma maemo e os principais módulos utilizados: GStreamer, ESD e a biblioteca ALSA.

7.1 Introdução

No início, os Internet Tablets não possuíam aplicações multimídia de qualidade para o usuário. A interatividade com o usuário era bastante limitada. Além disso, não havia bom suporte para os vários formatos de mídia: o usuário deveria instalar vários decodificadores para reproduzir os conteúdos multimídia. Para solucionar esses problemas, criou-se o *media center* Canola. Com uma interface que provê uma rica experiência para o usuário, o Canola também trouxe um suporte bem completo a vários formatos de mídia. Devido às inovações em interface gráfica e à qualidade da solução, o Canola agrega bastante valor ao Internet Tablet. Alguns usuários compram o Internet Tablet apenas para utilizá-lo como *media center*, o que é possível com o uso do Canola. Com isso, aplicações multimídia passaram a ser tornar bastante atraente para o usuário.

É importante que o desenvolvedor tenha bibliotecas de qualidade a sua disposição para o desenvolvimento de aplicações de qualidade. A plataforma maemo não possui nenhum mó-

dulo exclusivo da plataforma, exceto alguns drivers de dispositivos específicos de hardware. Todos os módulos também estão presentes nos desktops, por exemplo GStreamer e ESD.

7.2 Módulos Principais

Os principais módulos do subsistema de multimídia da plataforma maemo estão ilustrados na Figura 7.1. Perceba que a aplicação não se comunica diretamente com interfaces de mais baixo nível, por exemplo Xlib e ESD: o intuito é facilitar o desenvolvimento com o uso de soluções que abstraem funções mais complexas e de baixo nível. Embora também seja possível a utilização das bibliotecas ALSA e libesd, grande parte das aplicações multimídia da plataforma maemo são baseadas no framework GStreamer.

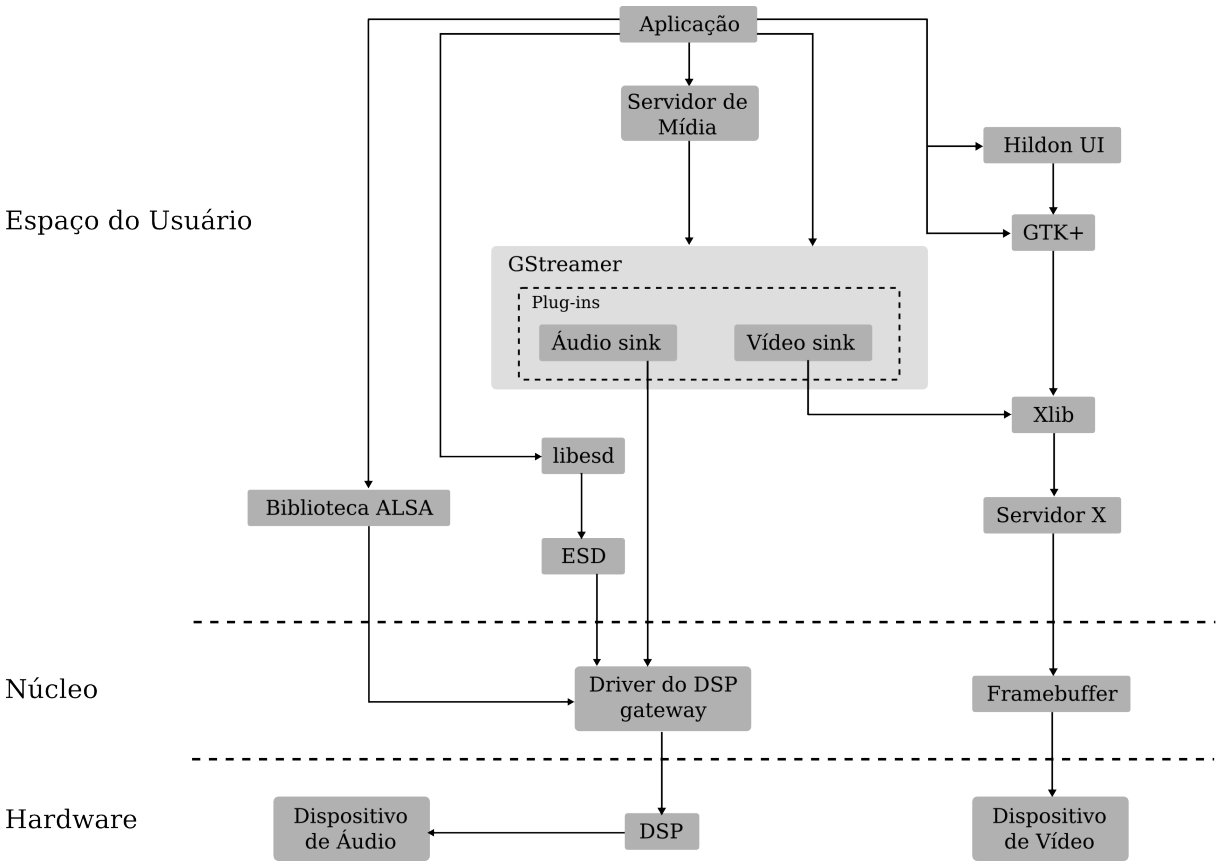

Figura 7.1: Principais elementos do módulo de multimídia da plataforma maemo.

7.2.1 ALSA

Advanced Linux Sound Architecture (ALSA) é um componente do núcleo do Linux utilizado para substituir o sistema Open Sound System (OSS) que provê drivers de dispositivos para placas de som. Alguns dos objetivos do projeto ALSA são a configuração automática do hardware responsável pelo áudio da plataforma e a gerência correta de vários dispositivos de áudio. Alguns frameworks existentes, tais como JACK, utilizam o ALSA para edição e mixagem de áudio, obtendo um resultado de boa qualidade.

ALSA foi desenvolvido para oferecer um conjunto de funcionalidades não suportadas pelo OSS, tais como operações de mixagem e síntese de áudio através do hardware. ALSA consiste de um conjunto de drivers de dispositivos do núcleo para várias placas de sons e também prover uma biblioteca chamada libsound. Os programadores devem utilizar a interface provida pela biblioteca (e não a do núcleo do Linux) para desenvolver suas aplicações para áudio. A biblioteca provê uma interface de programação de alto nível e de fácil utilização pelo programador, com um mecanismo de nomenclatura de dispositivos de áudio, abstraindo a complexidade dos detalhes de baixo nível, como os arquivos de configuração do dispositivo.

Por outro lado, drivers OSS são criados a nível de chamadas de sistema do núcleo e exigem que o desenvolvedor especifique detalhes do dispositivo de áudio. Para manter compatibilidade com versões anteriores, ALSA provê módulos do núcleo Linux que emulam controladoras de som baseados em OSS, de modo que a aplicação continue a execução sem problemas. Uma biblioteca que empacota a funcionalidade de emulação de controladoras OSS, chamada de libaosso, está disponível para emular a API OSS sem a necessidade de módulos do núcleo.

ALSA provê plug-ins, os quais permitem o suporte a novos dispositivos, incluindo dispositivos virtuais totalmente desenvolvidos em software. O sistema também provê um conjunto de ferramentas utilitárias baseadas em linha de comando, incluindo mixagem de som e reprodutor de arquivos de áudio.

7.2.2 ESD

Esound (ESD) é um daemon de som que abstrai a utilização dos dispositivos de som para os vários clientes existentes no sistema. Em outros sistemas Linux que utilizam o Open Sound System (OSS), a utilização do dispositivo de som é realizada por apenas um processo. Tal limitação não é aceitável em um ambiente desktop como o GNOME, no qual espera-se que várias aplicações utilizem o dispositivo de áudio (decodificadores, vídeo conferência, etc.). O daemon ESD se conecta ao dispositivo de áudio e realiza a conexão dos vários clientes, gerenciando corretamente os vários fluxos de som e enviando o resultado para o dispositivo. Para estabeler conexão com o ESD, é necessário que a autenticação ocorra com sucesso, para evitar que usuários não autorizados escutem o fluxo de áudio através do dispositivo. É possível também estabelecer conexões com clientes remotos. A comunicação de aplicativos ESD é ilustrada na Figura 7.2.

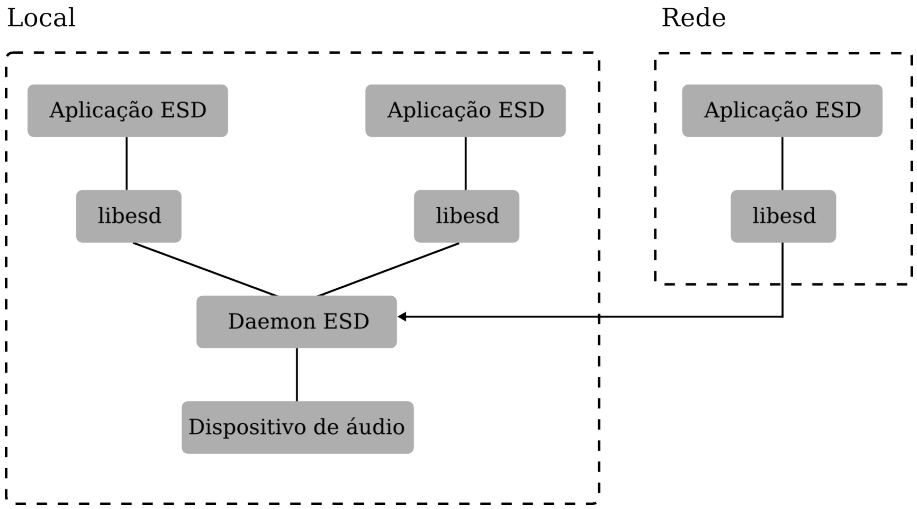

Figura 7.2: Comunicação entre aplicativos ESD.

As aplicações que desejam se conectar ao daemon ESD devem utilizar a biblioteca libesd. Assim como manipulação de arquivos, a conexão ESD precisa ser aberta primeiramente. Depois, o daemon ESD será automaticamente inicializado pela libesd se não houver um já presente no sistema. Os dados são então lidos ou escritos pelo daemon ESD. Para um cliente ESD que está sendo executado na mesma máquina na qual está presente um daemon ESD inicializado, os dados são transferidos através de um socket local, e depois escritos no dispositivo de áudio pelo daemon ESD. Para um cliente localizado em uma máquina remota,

os dados são enviados para a libesd na máquina através da rede para o daemon ESD. O processo é totalmente transparente para a aplicação.

7.2.3 GStreamer

O GStreamer é um framework desenvolvido para criar aplicações que manipulam fluxo (*streaming*) de mídia de alguma forma: conversores de formatos, reprodutores de mídia (vídeo e aúdio), gravadores de mídia (vídeo e aúdio), dentre outras possibilidades. É uma solução multi-plataforma, com suporte para Linux x86, PPC, ARM, Solaris x86 e SPARC, MacOSX, Microsoft Windows e IBM OS/400. Embora existam outras possibilidades para criação de aplicações multimídia, o GStreamer é a solução de fato na plataforma maemo. Na Figura 7.3 está ilustrada a arquitetura de uma aplicação multimídia maemo.

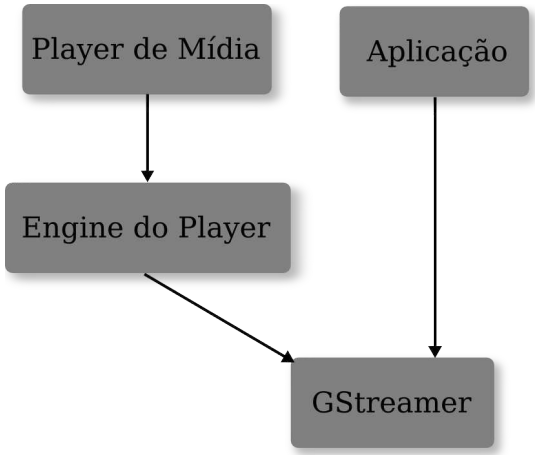

Figura 7.3: Arquitetura de uma aplicação multimidia para a plataforma maemo.

GStreamer possui uma série de componentes para construir um reprodutor de mídia para vários formatos, por exemplo, MP3, Ogg/Vorbis, MPEG-1/2, AVI, Quicktime e mod. Não se trata de apenas uma outra biblioteca de mídia: com a série de elementos, é possível uni-los em um único fluxo de execução e construir um editor de mídia, por exemplo. Com licença LGPL, o GStreamer passou a ser bastante utilizado por várias aplicações multimídia, por exemplo Amarok, Bonfire, Istanbul, Listen, Pitivi e o Canola.

O principal objetivo do GStreamer é oferecer uma infra-estrutura de multimídia de qualidade para o sistema Linux. Em outros sistemas operacionais (Windows e MacOS), há um

bom suporte para dispositivos multimídia, criação de conteúdo e processamento em tempo real. Os principais problemas relacionados com multimídia na plataforma maemo são:

Duplicação de código Para reproduzir um determinado arquivo, o usuário deve escolher um entre os vários reprodutores de mídia disponíveis no sistema Linux. Basicamente, tais reprodutores apenas reimplementam as mesmas funcionalidades (por exemplo, reprodução de arquivos MP3) e escrevem novo código. Assim, é necessário esforço e tempo para testar e depurar as funcionalidades de cada um dos reprodutores;

Transparência de rede Para utilização de funcionalidades que manipulam fluxo de mídia de rede, na maioria das vezes, o usuário deve conhecer qual protocolo deve ser utilizado e outros detalhes específicos de protocolo de rede. Tal problema pode ser um empecilho para utilização de aplicativos multimídia;

Inexistência de mecanismos para unificação Vários reprodutores/bibliotecas já tentaram estabelecer mecanismos para unificação das funcionalidades entre os vários aplicativos multimídia. Contudo, nenhum desses mecanismos foi adotado pelos desenvolvedores.

Arquitetura

O GStreamer possui uma arquitetura baseada em plug-ins, ou seja, componentes que podem ser facilmente adicionados ou removidos do framework. Os plug-ins podem ser disponibilizados sob diferentes licenças (LGPL, BSD, etc.) e devido a sua quantidade, foram divididos em 4 conjuntos: ***base***, com plug-ins que provêem funcionalidades básicas; ***good***, os quais são plug-ins de boa qualidade e com licença LGPL; ***ugly***, que são plug-ins de qualidade mas com licenças que impõem restrições na distribuição; e ***bad***, os quais não são de boa qualidade e muitos não são mais mantidos. A arquitetura do GStreamer é ilustrada na Figura 7.4. É provida uma camada abstrata para a manipulação dos plug-ins já instalados, reutilizando funcionalidades já existentes e obtendo um tempo maior para melhorias do aplicativo.

No GStreamer, o processo de execução inicia-se em uma fonte (source), o qual pode ser um fluxo de mídia da rede, um arquivo no disco, o áudio do microfone, etc. O fluxo é sempre recebido por um sorvedouro (sink), que pode ser os auto-falantes, a tela ou um arquivo em disco. O GStreamer é composto basicamente por três estruturas: *elements*, *bins/pipelines* e *pads*. Todo componente do GStreamer é um *element*, exceto o fluxo de mídia. Um *element*

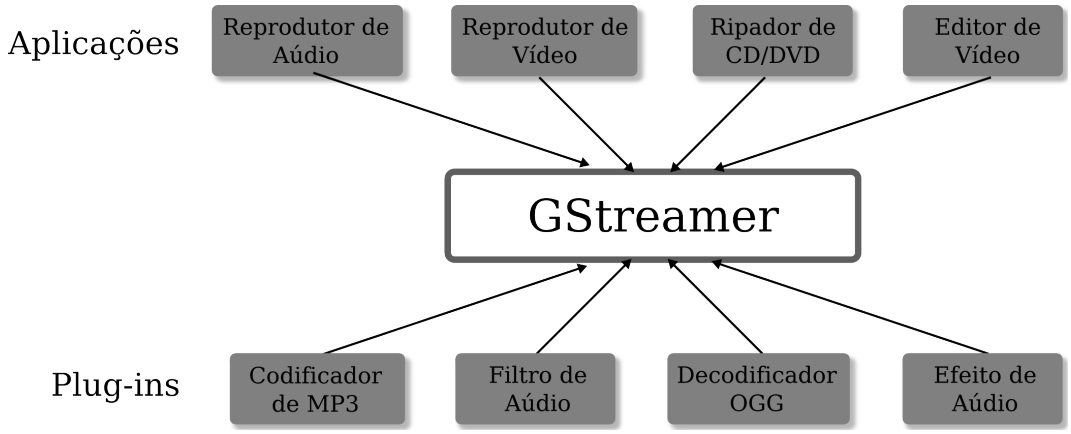

Figura 7.4: Arquitetura do framework GStreamer.

deve ter pelo menos uma porta de entrada (*source*), uma porta de saída (*sink*), ou as duas ou até mesmo vários de cada um. Um *element* pode estar em um dos seguintes estados: ***null***, o qual é o estado padrão; ***ready***, com buffers alocados e arquivos abertos, contudo com o fluxo de mídia em espera; ***paused***, estado cujo fluxo está aberto, porém seu movimento está parado e ***playing***, igual ao estado ***paused***, mas o fluxo está em movimento. A Figura 7.5 ilustra dois *elements* conectados e o fluxo em movimento.

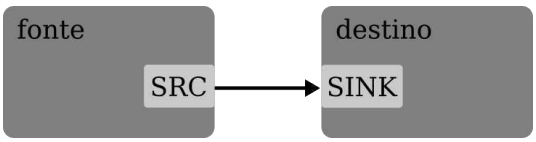

Figura 7.5: *Elements* e fluxo em execução.

Um *bin* é um contêiner de *elements*. Na Figura 7.6 está ilustrado um *bin* com alguns *elements* inseridos nele. Podemos também observar que no canto esquerdo da figura, encontra-se um *pad* (*sink*) ligado ao primeiro elemento. Trata-se de um *ghost pad*. Um *pipeline* é um tipo especial de *bin*, o qual além da adição de outros *elements*, também permite a execução desses *elements* nele inseridos.

No código abaixo, está descrito como criamos dois elements (um `gnomevfssrc` e um `dspmp3sink`) e depois os adicionamos em um *pipeline* para serem posteriormente executados.

```
1 #include <gst/gst.h>
```

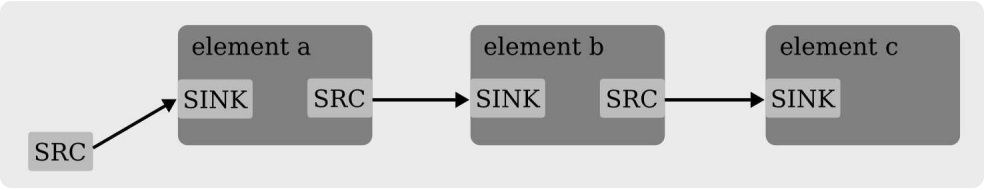

Figura 7.6: *Elements* contidos em um bin.

```
2
3  int main(int argc, char** argv) {
4      GstElement* pipeline;
5      GstElement* src, sink;
6
7      pipeline = gst_pipeline_new("pipeline");
8
9      src = gst_element_factory_make("gnomevfssrc","source");
10     g_object_set(G_OBJECT(src), "location", "/home/user/MyDocs/.sounds/
           A_Sample.mp3", NULL);
11
12     sink = gst_element_factory_make("dspmp3sink","sink");
13
14     gst_bin_add_many(GST_BIN(pipeline),src,sink,NULL);
15     gst_element_link(src,sink)
16 ...
17 }
```

```
1  import pygst
2  pygst.require("0.10")
3  import gst
4
5  pipeline = gst.Pipeline("pipeline")
6
7  src = gst.element_factory_make("gnomevfssrc", "src")
8  src.set_property("location","/home/user/MyDocs/.sounds/A_Sample.mp3")
9  pipeline.add(src)
10
11 flt = gst.element_factory_make("dspmp3sink", "sink")
12 pipeline.add(sink)
```

```
13
14  ...
15  }
```

Todo *element* deve ter pelo menos um *pad*, os quais são portas pelas quais o fluxo passa. Através deles, podemos restringir o tipo de dados que entram ou saem do *element* de maneira bem específica. Por exemplo, se quisermos reproduzir um arquivo em formato OGG, é necessário definir *elements* corretos para tratar os dados. Além disso, os *pads* também oferecem informações sobre suas capacidades para *elements* que queiram se conectar com eles. Existem três tipos diferentes de *pads*: ***dinâmicos***, os quais são adicionados ou removidos conforme a necessidade; ***request***, que podem ser criados sob demanda e ***ghost pads***, os quais são *pads* de algum element em um *bin* que pode ser acessível diretamente do *bin*. Um *ghost pad* pode ser comparado com um link simbólico UNIX, tornando possível o acesso a *bins* como se fossem *pads* por outras partes do código.

O GStreamer provê uma ferramenta bastante útil para desenvolvimento: o `gst-inspect`, uma ferramenta de linha de comando que permite saber quais os plug-ins e elements estão instalados no sistema. Com a ferramenta, também otem-se informações detalhadas sobre algum *plug-in* ou *element* específico. Um exemplo de utilização da ferramenta `gst-inspect` é descrito a seguir.

```
/ $ gst-inspect gnomevfs
Plugin Details:
  Name:                 gnomevfs
  Description:          elements to read from and write to  Gnome-VFS uri's
  Filename:             /usr/lib/gstreamer-0.10/libgstgnomevfs.so
  Version:              0.10.14
  License:              LGPL
  Source module:        gst-plugins-base
  Binary package:       GStreamer Plugins (Ubuntu)
  Origin URL:           https://launchpad.net/distros/ubuntu/+source/gst-plugins-base

  gnomevfssrc: GnomeVFS Source
  gnomevfssink: GnomeVFS Sink
```

```
  2 features:
  +-- 2 elements

/ $ gst-inspect dspmp3sink
No such element or plugin 'dspmp3sink'
```

Observa-se que existem os elements **gnomevfssrc** e **gnomevfssink**, presentes no *plug-in* **gnomevfs**. Contudo, não existe o *element* **dspmp3sink**, utilizado no código. Caso esse código seja executado, um erro ocorrerá.

7.3 Conclusão

Aplicações multimídias de qualidade devem ser providas para a plataforma maemo com o intuito de agregar maior valor à linha de produtos Internet Tablet. Existem bibliotecas que auxiliam no desenvolvimento de aplicações dessa natureza e tornam mais fácil a criação de aplicativos com qualidade. Para desenvolvedores desktop Linux, o esforço no porte de aplicações multimídia já existentes se concentra principalmente na interface gráfica, uma vez que os módulos de multimídia são iguais ao da plataforma maemo (GStreamer, ESD, etc.). A principal ferramenta para criação de aplicações multimídia para a plataforma maemo é o framework GStreamer, o qual possibilita a reutilização de módulos já existentes, bem como a criação de novos plug-ins por parte do programador.

Capítulo 8

Conectividade

O principal objetivo da plataforma maemo é oferecer "conectividade sem limites" para os usuários. Ou seja, possibilitar a conexão com a Internet e tambem outros dispositivos presentes em uma rede através de suas interfaces de comunicação (Wi-Fi, Bluetooth e também WiMax). É importante que as aplicações explorem as capacidades de conectividade da plataforma e apresentem soluções interessantes para problemas de comunicação, bem como aplicativos que explorem ao máximo essa capacidade. Neste capítulo, a arquitetura do subsistema de conectividade da plataforma maemo é apresentada: os módulos principais bem como as bibliotecas mais utilizadas no desenvolvimento.

8.1 Introdução

O subsistema de conectividade da plataforma maemo foi desenvolvida obedecendo as convenções de sistemas Linux. Ele está presente na área de memória do usuário e é invocado pelo núcleo Linux através de bibliotecas C. O principal canal de conexão com a Internet é a interface WLAN, mas conexões *dial-up* através das redes de dados dos celulares também é possível. O único meio de conexão com o celular é o Bluetooth. O suporte de software da biblioteca Bluetooth é baseado no BlueZ o qual é a implementação *de facto* de Bluetooth para Linux. Os principais componentes de conectividade da plataforma maemo estão ilustrados na Figura 8.1.

UI de Conectividade maemo Módulos prontos de interface gráfica de conectividade. Incluem um gerenciador de conexões, um painel de controle e outros diálogos diferen-

tes;

Daemon de Conectividade maemo - ICd API LibConIC funciona juntamente com o ICd, sendo responsável pela gerência dos pontos de acesso Internet (*Internet Access Points* - IAPs). Esse componente é responsável por ambas conexões WLAN e Bluetooth;

OpenOBEX Interface para serviços OBEX. O principal usuário dessa biblioteca é o módulo OBEX gnome-vfs;

Empacotador OBEX Implementação do protocolo aberto Object Exchange (OBEX). Mais informações sobre o OpenOBEX pode ser encontrado em `http://triq.net/obex/`;

Pilha Bluetooth BlueZ Implementação de facto do Bluetooth para Linux. Mais informações sobre BlueZ podem ser encontrados em `http://www.bluez.org`;

API BlueZ D-Bus Biblioteca BlueZ que recebe comandos via D-Bus;

Daemon de Conectividade WLAN Daemon para controle das conexões WLAN;

Controladora de Dispositivo WLAN Controladora de dispositivo da interface Wireless LAN (IEEE 802.11g). A controladora é comporta por duas partes: um executável (parte fechada) e um módulo aberto que integra o executável ao núcleo Linux.

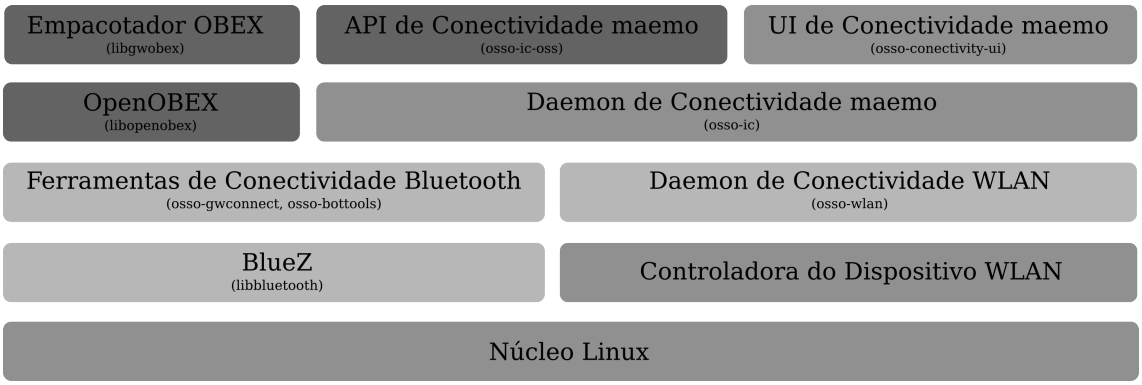

Figura 8.1: Elementos do subsistema de conectividade da plataforma maemo.

Um outro elemento importante no que diz respeito à conexão com Internet na plataforma maemo são os Pontos de Acesso a Internet (PAI). Tais elementos representam uma conexão

lógica com a Internet, o qual será definido pelo usuário de acordo com suas necessidades. Cada PAI possui um nome único e também define o canal de conexão (por exemplo, WLAN, CSD ou GPRS) a ser utilizado, bem como várias outras características: taxa de transferência de dados, login, senha e servidor proxy, por exemplo.

8.2 Subsistemas de conectividade

O subsistema de conectividade é composto basicamente por três elementos:

Gerente de Conexões Provê elementos de interface gráfica para gerência de conexões com o celular e com a Internet. Os principais elementos estão acessíveis a partir do Painel de Controle e da Barra de Status. Tais elementos também estão disponíveis em forma de serviços, podendo ser utilizados por outras aplicações. As principais responsabilidades desse componente são disponibilizar informações sobre as conexões atuais, atualizar os dados das conexões e provê diálogos para escolha, encerramento e mudança de conexão;

Acesso à Internet Possibilita a conexão com Internet através de celulares e interface WLAN. É responsável por prover mecanismos para gerência e configuração dos PAIs, oferecer uma API para estalecer conexões com a Internet através de WLAN e Dial-up/PPP [1] e também informar ao Gerente de Conexões sobre o status das conexões com a Internet;

Acesso ao Celular Permite estabelecer conexões Bluetooth com células de diferentes perfis. As principais responsabilidades são: busca por dispositivos e serviços Bluetooth, manutenção do registro do celular, informar ao Gerenciador de Conexões sobre o status das conexões com celulares e prover acesso fácil ao OpenOBEX.

O subsistema de Acesso à Internet também é responsável pela conexão com o celular, se necessário, usando os serviços do subsistema de Acesso ao Celular. Se alguma aplicação

[1] Protocolo de comunicação comumente utilizado para estabelecer conexão direta entre dois nodos utilizando um cabo serial, uma linha telefônica, um telefone celular, um canal Bluetooth ou links de fibra ótica, por exemplo.

necessitar de obter acesso aos arquivos localizados então um seletor de arquivos irá consultar o subsistema de Acesso ao Celular.

Durante o modo *offline*, as interfaces WLAN e Bluetooth não estão ativadas. A Entidade de Gerência do Sistema do Dispositivo da plataforma maemo provê informação sobre as transições a partir de e para o modo *offline*. Conexões à Internet e com o celular são bastante diferentes em termos de natureza e comportamento. As próxima sessões trazem detalhes maiores sobre cada um desses elementos.

8.2.1 Acesso ao Celular

Este é um subsistema que gerencia conexões com celulares. Possui mecanismos utilitários para encontrar celulares próximos e verificar quais os serviços que estão disponíveis. Essa busca de serviços é implementada da mesma maneira que o padrão Bluetooth. O subsistema de Acesso ao Celular também mantem um registro (através do GConf) com todos os celulares que estiveram conectados com o Internet Tablet em algum momento, e também provê uma lista com esses contatos para que o usuário possa escolher posteriormente. O subsistema de Acesso ao Celular é baseado na implementação das especificações Bluetooth para Linux, o framework BlueZ, o qual oferece uma interface de socket para os protocolos L2CAP e HCI para aplicações que executem no modo usuário.

A princípio, qualquer celular que suporte o Protocolo de Descoberta de Serviços Bluetooth (Bluetooth Service Discovery Protocol - SDP), perfil de Rede Dial-Up (Dial-up Networking - DUN) [2] e Perfil de Transferência de Arquivos (File Transfer Profile - FTP) pode ser conectado à plataforma maemo. Entretanto, há variação especialmente no nível de serviços para transferência de arquivos e OBEX nos diferentes celulares. Alguns produtos limitam o acesso via sistemas de mensagem (Object Push), enquanto alguns mais sofisticados disponibilizam o conteúdo do cartão de memória e galeria de imagens. Celulares mais recentes possuem suporte OBEX, e podem ser utilizados para obter informações mais específicas sobre o sistema de arquivos do celular.

Maemo se conecta ao celular sob demanda, ou seja, quando uma aplicação requisita

[2] Representa a obtenção de uma conexão wireless com a Internet de um computador (desktop ou laptop) usando um celular com Bluetooth habilitado. DUN Bluetooth é uma boa alternativa para Internet *wireless* onde não há pontos de acesso wireless disponíveis para oferecer um acesso mais eficiente e rápido.

uma conexão. Por exemplo, quando um browser para Internet está prestes a abrir uma URL, requisita-se ao subsistema de Acesso a Celular que uma conexão com o celular seja estabelecida. Isto faz com que o subsistema de Acesso a Celular conecte um dispositivo RFCOMM a um determinado serviço (neste caso, DUN) no celular. De maneira similar, o seletor de arquivos pode determinar uma outra conexão para transferência de arquivos para o celular usando um outro dispositivo RFCOMM. Após se conectar com o serviço, a aplicação em questão pode abrir um dispositivo RFCOMM local. O acesso ao seletor de arquivos é realizado através da camada do GnomeVFS com o intuito de obter um acesso transparente com o celular da mesma maneira que a memória flash interna e cartões de memória são acessados.

8.2.2 Acesso à Internet

O subsistema de Acesso à Internet gerencia conexões com a Internet através da interface WLAN e o celular. Também é responsável pela configuração e gerência de PAIs. O Acesso à Internet provê serviços para conexão TCP/IP. Podem ser estabelecidas com:

- Conexões WLAN com um ponto de acesso *wireless*;
- Conexões Bluetooth através do celular usando o PPP e um modem móvel (do celular).

8.3 Daemon de Conectividade maemo - ICd

O Daemon de Conectividade maemo (*Maemo connectivity daemon* - ICd) estabelece conexões com a Internet através do Bluetooth e WLAN. Ele também é responsável por iniciar alguns serviços, tais como DHCP e PPP. Esta sesão descreve como o ICd funciona internamente. As próximas subseções descrevem o comportamento e os aspectos internos desses componentes em detalhes, bem como as interfaces que eles implementam.

8.3.1 Decomposição

Quando o ICd recebe uma requisição para ativar ou desativar um PAI, é possível ocorrer um dos seguintes cenários: i) o ICd ativará o PAI ou ii) caso nenhum tenha sido selecionado como padrão, será disponibilizado um diálogo para requisitar ao usuário que seja escolhido

um PAI conforme ilustrado na Figura 8.2. Depedendo do tipo do PAI (celular ou wireless), o ICd ativará ou desativará ou o DUN (*Bluetooth dial-up networking*) Bluetooth ou o a interface WLAN.

Figura 8.2: Diálogo para escolha de um PAI.

O ICd consegue determinar quais aplicações solicitam um PAI, pois armazena os seus nomes na base de serviços D-Bus. Tal fato permite ao ICd detectar situações nas quais processos que utilizavam um PAI foram abortados ou apresentaram falha. ICd também implementa um mecanismo de timeout que finaliza um PAI ativo quando nenhum pacote foi enviado durante um determinado espaço de tempo. Esse parâmetro do sistema é configurável.

Na versão 3.0 da plataforma maemo, o daemon de Conectividade à Internet trouxe a funcionalidade de criação automática de conexões. Em outras palavras, o Internet Tablet tentará estabelecer conexões com os PAIs já previamente armazenados no sistema e tentará manter a conexão o maior tempo possível, exceto se a conexão ficar inativa por um determinado tempo. Com essa funcionalidade, aplicações como caixa de e-mails e leitor RSS estarão sempre atualizados. O dispositivo também sempre estará on-line para uso, por exemplo, para receber chamadas VoIP ou iniciar um chat. Nas primeiras versões, a conexão com a Internet era automaticamente encerrada quando nenhuma outra aplicação a utilizava, ou quando a conexão se apresentar inativa por um determinado período previamente estabelecido através do parâmetro *connection timeout*.

Quando não estiver conectado, o dispositivo busca por todos os PAIs salvos e tenta se conectar automaticamente, levando em consideração o parâmetro para determinar qual o tempo de busca por um novo PAI, conforme ilustrado na Figura 8.3: 5, 10, 30 ou 60 minutos. Todos os outros valores serão automaticamente modificados para “Never". Neste caso, a funcionalidade de conexão automática é “desligada". Assim, o dispositivo se comportará como nas primeiras versões: as conexões só serão estabelecidas quando requisitadas pela aplicação.

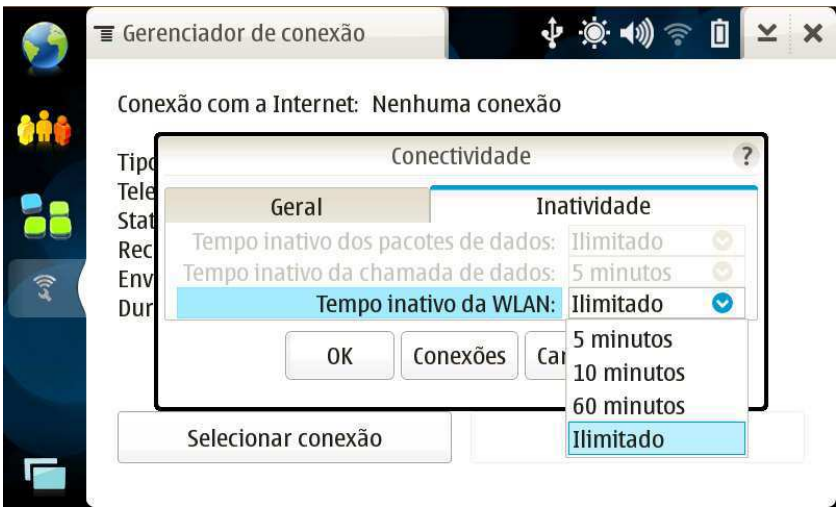

Figura 8.3: Configuração de parâmetros do subsistema de conectividade.

O ICd também é responsável pela criação da conexão. É de responsabibilidade de cada aplicação manter os seus dados atualizados e então provê a funcionalidade *always-online*. Ao implementar uma aplicação que utilize o sistema ICd, os seguintes aspectos devem ser considerados:

- A aplicação deve sempre utilizar a conexão disponível;
- Assim como era realizado nas versões anteriores, se o dispositivo não está conectado e uma conexão é requisitada através da interação do usuário, a aplicação deve requisitar a criação de uma conexão usando a API LibConIC;
- O usuário deve ser informados sobre as atualizações, tornando possível quando os dados de conexão são modificados;

- A aplicação deve se registrar via LibConIC e ouvir os sinais emitidos pelo ICd (conexão criada, perdida ou modificada) e reagir da seguinte maneira:
  - Conexão criada: usa a conexão e atualiza todos os dados;
  - Conexão perdida: modifica para o estado em espera até que uma nova conexão seja criada;
  - Conexão modificada: usa uma nova conexão.
- Atualizações automáticas de dados são executadas em background e de maneira transparente:
  - Deve-se evitar repassar avisos ao usuário através de banners desnecessários e diálogos;
  - *Logins* e senhas devem ser salvos para que atualizações automáticas sejam realizadas sem a necessidade de *prompts*;
  - Neste caso, falhas não podem exibir diálogos de notificação de erros.
- A infraestrutura de conectividade gerencia situações de erros de maneira centralizada.

A criação automática de conexões pode ser também “desligada"usando o modo *offline*. Neste modo, o parâmetro de configuração que habilita a WLAN pode ser modificado. Dependendo do estado do parâmetro de configuração, os PAIs WLAN podem estar ou não habilitados. Conexões Bluetooth são normalmente desabilitados no modo *offline*.

8.3.2 Bluetooth Dial-up Networking

ICd utiliza PPP para estabelecer conexões através de interfaces Bluetooth DUN. Se houver algum PAI ativo usando um DUN Bluetooth, o antigo PAI é primeiramente desativado. O PAI é ativado da seguinte maneira:

- O dispositivo utilizado pelo DUN Bluetooth é adquirido a partir do `btcond` [3]. Se o dispositivo não estiver disponível por alguma limitação (várias conexões Bluetooth já existentes, por exemplo) o ICd disponibiliza uma mensagem de erro para o usuário e aborta com uma mensagem para o D-Bus;

[3] *Bluetooth Connection Daemon*

- O ICd inicializa o PPP usando o conjunto de chamadas de sistema *exec*. Ele direciona o PPP a utilizar o dispositivo DUN Bluetooth obtido com os parâmetros da configuração dial-up para o tipo especificado de PAI DUN. Se o PPP não conseguir obter a conexão estabelecida, o ICd mostrará uma mensagem de erro para o usuário e abortará com uma mensagem D-Bus. Quando a conexão PP é estabelecida, scripts específicos do PPP são executados, os quais configuram o IP dinâmico e enviam uma mensagem de mudança D-Bus para todas as aplicações interessadas para indicar que um PAI foi estabelecido.

Se o PAI ativo não estiver utilizando um DUN Bluetooth, ele será encerrado após estabelecer uma conexão PPP. O DUN Bluetooth é encerrado através do envio de sinais SIGINT e SIGTERM para o daemon PPP. Além de finalizar o daemon PPP, todas as entradas de rotas associadas à interface dial-up PPP serão encerradas. Alguns scripts PPP removem as configurações relacionadas à conexão com IP dinâmico e enviam uma mensagem D-Bus de mudança de estado notificando a desativaçào do PAI.

8.3.3 WLAN

Quando se conecta com a WLAN, o ICd precisa se associar a uma rede e habilitar a autenticação EAP e também o cliente DHCP quando necessário. Independentemente se há um PAI ativo utilizando o WLAN, a rede WLAN requisitada será antes verificada para assegurar se está disponível. O PAI atual pode ser desativado se a rede requisitada for encontrada e o PAI atual estiver usando a WLAN. WLAN é ativado da seguinte forma:

- Caso a rede necessite de autenticação, então o procedimento de autenticação EAP é inicializado. Durante esse processo, alguns diálogos de interface gráfica são disponibilizados para o usuário como parte da autenticação. Quando o processo de autenticação EAP for finalizado, as chaves de segurança serão modificadas para a rede WLAN e mensagens para mudança de estado são enviadas pelo `wlancond` [4]. O ICd recebe tais mensagens, mas as ignora e espera por uma resposta da autenticação EAP. Se a autenticação EAP falhar, o ICd aborta com uma mensagem de erro D-Bus;

[4] *WLAN Connection Daemon*

- Após o ínício do processo de autenticação EAP, o ICd solicita que o `wlancond` se associe com uma rede WLAN. Configurações relacionadas às chaves de segurança previamente estabelecidas são fornecidas neste momento. Se uma conexão com a rede WLAN não poder ser estabelecida, o ICd aborta com um erro;

- Um cliente DHCP é um programa que executa em modo *standalone*. Dessa forma, ele é inicializado usando chamadas ao sistemas `exec` quando os PAIs WLAN solicitam a aquisição de endereço IP dinâmico. Quando o cliente DHCP obtem um endereço IP, os parâmetros relacionados à rede TCP/IP são configurados e uma mensagem D-Bus é enviada pelo ICd. Se o endereço IP não for obtido, o ICd encerra o cliente DHCP e aborta a operação com uma mensagem de erro D-Bus.

8.3.4 Biblioteca LibConIC

API de conectividade à Internet (LibConIC) é uma biblioteca para aplicações com o intuito de gerenciar conexões à Internet nos dispositivos maemo. Foi inicialmente utilizada na primeira versão do OS 2007, substituindo a antiga biblioteca OSSO IC API (osso-ic-lib), a qual foi completamente removida na versão OS 2008. A documentação completa da API pode ser encontra nas especifações de bibliotecas da plataforma maemo [5]. A Libconic é uma biblioteca de alto nível e orientada a objetos, sendo usada para:

- Requisitar conexões com a Internet;

- Ouvir eventos do estado de conexões com a Internet;

- Receber estatísticas sobre as conexões com a Internet;

- Obter configurações de proxy da conexão corrente;

- Obter a lista de conexões salvas pelo usuário (PAIs).

8.3.5 Bibliotecas Bluetooth

Esta seção descreve como as bibliotecas maemo funcionam internamente. As próximas subseções explicam o comportamento e decomposição dos componentes de Bluetooth em deta-

[5] http://maemo.org/api_refs/4.0/libconic/index.html

lhes.

Libgwobex

Libgwobex provê acesso às funcionalidades da biblioteca libopenobex através de uma interface para empacotamento de funções.

Libopenobex

A biblioteca LibOpenOBEX implementa uma sessão de procotolo OBEX genérica, e não o Framework de Aplicações OBEX. OBEX é um protocolo criado para possibilitar a troca de dados entre direferentes tipos de conexão (por exemplo, Bluetooth, IrDA). Informações específicas sobre o protocolo OBEX podem ser encontrada em `http://www.irda.org`. OBEX é um protocolo similar ao HTTP, exceto pelas seguintes diferenças:

Transporte HTTP normalmente funciona sobre uma conexão TCP/IP. Por outro lado, OBEX é comumente utilizado sobre uma pilha IrLAP/IrLMP/Tiny TP de um dispositivo IrDA, enquanto que funcionado com o Bluetooth, o OBEX está implementado sobre uma pilha em Banda Base/Link Manager/L2CAP/RFCOMM;

Transmissões em dados binários HTTP utiliza um formato mais legível para transmissão de dados, enquanto que o oBEX utiliza dados em formato binário chamado de headers para troca de informações sobre uma solicitação ou um objeto. Estes são mais simples de se implementar para dispositivos com características limitadas;

Suporte de sessões As transações HTTP não possuem armazenam estado. Geralmente, um cliente HTTP estabelece uma conexão, efetua somente um serviço, recebe uma resposta e encerra a conexão. No OBEX, uma só conexão de transporte pode ser utilizada para várias operações. É possível armazenar o estado de uma conexão, inclusive se ela foi finalizada inesperadamente.

BlueZ

BlueZ provê implementação ea especificação do Bluetooth (camadas e protocolo) para Linux. É flexível, eficiente e utiliza uma implementação modular. Possui várias funcionalidades interessantes, tais como: implementação modular, processamento *multithreaded*

de dados, suporte para vários dispositivos Bluetooth, abstração de hardware, interface socket padrão para todas as camadas e suporte a segurança a nível de serviço e dispositivo. A biblioteca funciona em várias plataformas, sendo suportada desde sistemas embarcados, até mesmo arquiteturas *multicore* e *hyper threading* (Intel x86, SUN SPARC 32/64bit, PowerPC 32/64bit e Intel StrongARM, por exemplo).

8.4 Conclusões

A plataforma maemo oferece um sistema bem completo para possibilitar a conexão de aplicações com a Internet ou com outras redes através de WLAN ou Bluetooth. Dessa forma, é importante que os softwares desenvolvidos explorem as possibilidades oferecidas pela plataforma e desenvolvam soluções eficazes para comunicação entre diversos elementos de uma rede. Além das interfaces Wi-Fi (IEEE 802.11) e Bluetooth, o modelo N810 WiMax trás uma nova possibilidade de conexão: a tecnologia WiMax, a qual permite maior taxa de transmissão de dados e dimunui os custos da infraestrutura de banda larga para conexão com o usuário final. Contudo, o suporte para utilização da interface WiMax ainda não está disponível para os desenvolvedores.

Capítulo 9

Segurança

Segurança é um requisito pouco considerado no desenvolvimento de aplicações embarcadas, uma vez que muitos desenvolvedores acreditam que dispositivos móveis não estão suscetíveis a ataques. Um dispositivo móvel guarda informações pessoais bastante importantes do usuário, tais como lista de contatos, mensagens de texto e e-mails. Além disso, qualquer ataque à alguma aplicação que o usuário usa constantemente (agenda, por exemplo) ou mesmo problemas na infra-estrutura (geração de tráfego de rede inútil), pode comprometer seriamente não somente o software, mas o sistema como um todo. Este capítulo apresenta informações suficientes para auxiliar os desenvolvedores em questões relativas à segurança de aplicações, principalmente projetos *open source*, discutindo fontes potenciais de falhas e situações de vulnerabilidade da aplicação. Neste capítulo, não se pretende produzir uma lista definitiva de tudo o que deve ser considerado ao se desenvolver software seguro para dispositivos móveis, tampouco descrever todos os possíveis ataques que podem ocorrer. Para maiores detalhes, o leitor pode consultar [42; 78; 89].

9.1 Elementos de Desenvolvimento de Software Seguro

9.1.1 Considerações sobre Segurança de Software

Segurança em software pode ter várias definições, sendo algumas baseadas em funcionalidades do sistema enquanto outras são definidas sobre seu real uso. Uma das definições que

melhor definem segurança em termos de desenvolvimento de software é aquela baseada em qualidade: aquele que consegue cobrir todas as necessidades para o qual foi criado é considerado de alta qualidade. De maneira similar, um software que também executa todas as funcionalidades esperadas sob ataque ou em um ambiente hostil é considerado seguro.

Essa definição já traz alguns dos principais conceitos de software seguro: i) os desenvolvedores de sistemas precisam entender o ambiente hostil no qual o software será implantado para saber como criar mecanismos de defesa e ii) o sistema precisa ser devidamente testado em um ambiente com as mesmas condições do ambiente hostil esperado.

9.1.2 Um Processo de Desenvolvimento Adaptado

Existem alguns modelos diferentes para o desenvolvimento de software seguro. Um dos mais bem documentados é o SDL (*Secure Development Lifecycle*) da Microsoft [42]. O número de passos a serem adicionados no ciclo de desenvolvimento varia bastante. Entretanto, ao se comparar as várias metodologias de desenvolvimento de software seguro em empresas, três fases parecem ser comuns em toda a indústria: análise de ameaças, implementação de modelos de segurança e teste de segurança. Desses passos, a análise de ameaças e o teste de segurança serão discutidos do ponto de vista do dispositivo embarcado. A implementação de modelos de segurança é também importante, porém é melhor apresentada em outras fontes da literatura, tais como [34; 42; 89].

9.2 Análise de Ameaça

9.2.1 Análise de Ameaça como Base de Software Seguro

Ataque e defesa de informações são exercícios constantes no desenvolvimento de software seguro. A fonte do ataque pode escolher qual o alvo, enquanto o sistema de defesa deve se defender de todos os demais ataques. Além disso, nem sempre a segurança é levada em consideração pelos usuários: algumas tarefas, como informação de senha e login, são realizadas sem nenhuma preocupação. Como conseqüência desses fatos, um sistema que não foi concebido seguindo certos quesitos de segurança está bastante funerável a ataques a partir de várias fontes. Para tornar o sistema mais seguro, é necessário realizar um tratamento

de ameaças, o qual é a base para qualquer trabalho que envolva engenharia de segurança.

A análise de ameaças possui dois objetivos principais: i) determinar o nível de segurança através da identificação do maior número possível de aspectos do sistema e mecanismos de defesa e ii) levar os participantes a pensarem como um invasor, a partir da maneira como a análise de ameaça é conduzida.

9.2.2 Como Conduzir Análise de Ameaças

Há diversas maneiras de realizar análise de ameaças. Uma das mais formais é mapear todo o fluxo de dados de uma aplicação usando diagramas arquiteturais que descrevem todas as interfaces e componentes. Essa tarefa se torna mais simples quando a arquitetura é bem projetada e também quando o design é baseado em observações realizadas em ataques anteriores que foram causados pela falta de um tratamento adequado às entradas lidas pela aplicação. Através do acompanhamento da passagem dos dados através dos vários componentes da aplicação (utilizando o diagrama citado anteriormente), pontos passíveis de ataques podem ser identificados.

Entretanto, esse tipo de análise de fluxo de dados não deve ser utilizada quando um sistema é atacado por outros meios além de entradas maliciosas, por exemplo, sujeitando a aplicação a executar em um ambiente diferente daquele na qual ela foi criada para ser executada. É possível utilizar outros métodos mais tradicionais como obter vários cenários de execução nos quais pode haver um ataque ao sistema, ou seja, um *brainstorming* para definir quais os possíveis ataques.

Brainstorming também é um bom método para induzir o pensamento de invasores da equipe de desenvolvimento. A partir de várias perguntas “mas e se...”, é possível determinar diversas maneiras (muitas já conhecidas, outras totalmente não consideradas anteriormente) de como atacar o sistema em análise. Para um profissional experiente na área de segurança de sistemas, é possível mapear todos os possíveis cenários de ataques. Porém, para a maioria dos profissionais leigos no assunto, o suporte em grupo tem apresentado bons resultados em relação à qualidade da análise de ameaças.

Cenário de Tratamento de Ameaças em Código *Open Source*

Acredita-se que grande parte dos desenvolvedores de projetos *open source* se preocupam com a segurança da aplicação. Contudo, do ponto de vista do software proprietário ou fechado, é importante a discussão de ameaças específicas em potencial que o desenvolvimento de projetos *open source* pode trazer. Deve-se observar que tais problemas não são ameaças de segurança de fato, mas em muitos casos, modificam a maneira como questões de segurança são implementadas.

Primeiramente, algumas licença *open source* definem que quaisquer mudanças realizadas no código fonte devem ser disponibilizadas com a mesma licença. Ao considerar correções de segurança, muitas modificações tornam públicas. Dessa forma, é muito fácil comparar as correções realizadas com a versão mais antiga e determinar qual o problema de segurança. Ao utilizar o código *open source*, questões de segurança estarão automaticamente expostas nas próximas versões do código fonte.

Em segundo lugar, muitos projetos *open source* não são levados a sério pelos próprios desenvolvedores. Dessa forma, em muitos deles ainda carecem de qualidade e segurança. Além disso, muitas correções de segurança deveriam ser implemenadas, porém outras mudanças são priorizadas no ciclo de manutenção no software *open source*. Assim, se um determinado software depende de serviços específicos a nível de segurança que precisam se corrigidos, dependendo do projeto *open source*, é necessário que o próprio desenvolvedor esteja pronto para realizar essas mudanças no projeto. Tal cenário também é considerado por licenaş *open source*, as quais geralmente lançam o código sem nenhum tipo de garantia.

Por último, o código aberto facilita a busca por partes vulneráveis no software. Em um código fechado, a busca por falhas de segurança é baseada em tentativa e erro, enquanto que em softwares *open source*, problemas de segurança estão bastante claros. Tais fatos não tornam projetos *open source* inseguros, mas exigem um tratamento diferenciado. Projetos *open source* também trazem benefícios em relação à segurança: ao se tornarem bastante utilizado, as chances de profissionais de segurançã explorarem o código e realizarem correções tendem a aumentar. Além disso, o código disponível permite que o próprio desenvolvedor realize as correções de segurança e liberá-las junto com o projeto em desenvolvimento, sem ter que esperar pelas próximas mudanças.

Segurança em Dispositivos Móveis

Conforme discutido anteriormente, dispositivos móveis diferem de computadores desktop e servidores. Embora essa diferença seja mencionada várias vezes e muitas considerações não sejam mais verdade daqui a alguns anos (principalmente problemas relacionados à limitação de recursos), certos fatores que diferem os dispositivos móveis dos não-móveis continuarão por muito tempo, e devem ser considerados ao se pensar em ameaças à segurança. Três categorias de ameaças relativas a mobilidade serão brevemente discutidas:

- Ameaças relativas aos portadores de dados móveis;
- Ameaças relativas a sistemas limitados;
- Ameaças relativas a natureza "pessoal" do dispositivo.

O canal de comunicação de um dispositivo móvel é muito menor do que o de um servidor ou um computador desktop. Mesmo com o surgimento de redes 3G HSPA, WiMAX e Wi-Fi que prometem taxas de trasmissão de vários megabits por segundo, não há nenhuma garantia que alcancem esse velocidade. Em áreas que não são cobertas por tecnologias de transmissão rápida, as taxas de transmissão bem baixas (próximas às velocidades de modems). Do ponto de vista do invasor, canais com banda pequena são mais fáceis de serem congestionadas usando um ataque de negação de serviço (*denial of service*).

Outro aspecto relacionado com os canais de comunicação é o modelo de tarifação. Muitos dispositivos móveis isolam a aplicação em execução a partir de dados específicos do portador. A transferência de dados pode ocorrer através de uma rede Wi-Fi ou do celular (Bluetooth) e o portador pode até mesmo se modificado durante uma determinada operação (por exemplo, requisições HTTP subseqüentes podem ser transferidas através de vários portadores). Portanto, a aplicação não pode determinar se o portador que está sendo utilizado transmite a uma taxa fixa ou variada. A tarifação varia bastante entre operadoras (quantidade de pacotes, quantidade de dados, etc.). O fato da aplicação ser atacada e levada a gerar muito tráfego de rede trás um impacto bastante negativo na conta do usuário e também em sua satisfação.

O fato do portador poder ser modificado de maneira dinâmica trás alguns desafios aos serviços de rede que estão sendo utilizados. Conexões com celulares e mensagens de texto

são consideradas seguras por utilizarem redes fechadas que são tratadas pelas operadoras. Entretanto, um portador pode modificar de repente e acessar uma rede Wi-Fi aberta, onde todos os pacotes podem ser acessados assim como portas que estão esperando por dados de entrada. A conexão segura deve ser devidamente concebida com um mínimo de segurança em mente.

A segunda maior diferença em relação aos dispositivos móveis é a fonte de energia. Historicamente, a tecnologia de bateria se desenvolveu mais lentamente do que outros elementos de dispositivos móveis (memória, processamento). Gerência de energia para dispositivos móveis é uma área complexa, e um programa de comportamento problemático pode gerar consumo desnecessário da bateria. Embora não seja considerado diretamente uma questão de segurança, deve-se considerar que uma bateria pode causar um erro de negaçao de serviço a todo o sistema.

Essa categoria de desafios relacionados à mobilidade também se aplicam a outros recursos além de energia. Processadores de dispositivos móveis geralmente executam em velocidades menores, possuem uma memória menor, menor capacidade de armazenamento e tela menor do que máquinas desktop ou servidores. Um ataque contra qualquer um desses recursos limitados pode também causar um erro de negaçao de serviço.

A terceira categoria está relacionada com o fato de dispositivos móveis serem de uso pessoal. Por estarem na mesmo localização geográfica do usuário, é possível realiza um ataque de software para identificar o paradeiro do usuário de forma exata. Além disso, os usuários guardam muitos dados pessoais no dispositivo, tais como lista de contados, agenda, notas pessoais, fotos, e-mails e mensagens de texto, por exemplo. O vazamento dessas informações constituem um problema de privacidade e a remoção desses dados pode causar descontentamento do usuário.

Essa característica pessoal dos dispositivos móveis é destacada pelo fato que um identificador único e estático é utilizado. Um dispositivo móvel geralmente possui um ou mais endereços MAC (para WLAN, Bluetooth e WiMAX), um IMEI (número serial do dispositivo GSM/3G) e um IMSI (identificador de inscrição GSM/3G). Todos esses identificadores são únicos e estáticos, possibilitando a identificação do usuário e dando informações suficientes para ataques. O software deve se preocupar com questões relativas à privacidade do usuário e restringir o uso de tais identificadores.

9.3 Teste de Segurança

9.3.1 Teste de Robustez

Uma vez que foi realizada a análise de ameaças e todas as precauções ao se implementa um código seguro, a próxima fase se resume aos testes. Parte dos testes são "positivos", ou seja, a aplicação é sujeita a um conjunto de casos de sucesso. Outros casos podem ser "negativos", nos quais a implementação é claramente para que aplicação falhe.

Provavelmente o teste mais importante de segurança é realizado através de um tipo específico de teste "negativo". Geralmente chamado de *teste de robustez*, este teste busca fazer com que a implementação falhe, ou seja, comporte-se de uma maneira que não seja robusta. Considerando a definição de segurança vista anteriormente, é necessário que um sistema com qualidade também seja seguro: em um ambiente propício a ataques (o que é realizado no teste de robustez), o sistema deve continuar em perfeito estado de utilização (ou seja, ainda robusto).

Geralmente, o teste de robustez se concentra na entrada disponível. Qualquer sistema que interaja com um ambiente, possui interfaces de entrada. Tipicamente, o que há são, por exemplo, parsers de protocolos, de arquivos, elementos de interface gráfica com o usuário, interfaces de programação, interfaces para chamada remota a procedimentos. A interface não é necessariamente utilizada pelo usuário. Por exemplo, mesmo o DNS, o qual retorna a algo a partir de um endereço web, é uma interface geralmente utilizada entre máquinas. Portanto, qualquer dado de entrada deve ser considerado suspeito e deve ser devidamente testado.

O teste de todo o sistema nem sempre pode ser considerado devido à falta de recursos. A prioridade deve ser dada às entradas que não podem ser autenticadas (o que significa que não há como determinar de onde veio) e também àquelas que são externas ao sistema. Por exemplo, as propriedades de robustez de uma API do sistema são testadas com uma prioridade menor do que a de um parser de arquivo que o usuário utiliza para ler um arquivo de entrada, e por sua vez o parser de arquivo tem prioridade menor do que o de um daemon de rede que observa uma porta de conexão e processa as requisições vindas de um computador remoto desconhecido. A análise de ameaças deve observar tais interfaces e determinar de que maneira elas devem ser testadas.

Teste de robustez é tipicamente realizado a partir das entrada que não são devidamente bem formadas (well-formed). Métodos diferentes de teste de robustez incluem *fuzzing*, ou seja, entradas válidas são obtidas de um gerador randômico. Os dados podem ser gerados de vários níveis: desde dados em formato binário até dados de mais alto nível, tais como estrutura de dados. Por exemplo, um fuzzer XML pode adicionar elementos XML não especificados e adicionar a um arquivo com elementos XML definidos.

Outra maneira de produzir casos de testes de robustez é através da geração de cenários com erros a partir das especificações da entrada. Por exemplo, se uma especificação diz que uma entrada de texto pode ter 16 bits de extensão, os casos de testes tentarão com tamanhos extremos: 0 e 17 bits. Esse tipo de síntese de casos de testes é mais eficiente que a geração randômica, mas necessita inicialmente de esforço extra, pois requer um bom conhecimento das falhas de cada parser.

Bugs encontrados com o auxílio de teste de robustez geralmente causam problemas sérios ou, no mínimo, baixa qualidade na usabilidade. Qualquer entrada com um conteúdo problemático pode causar uma pertubação na utilização normal; em muitos casos, o melhor a se fazer é descartar os dados com problemas, e algumas vezes reportar o usuário com notificações. Vale salientar que os testes de conformidade geralmente realizados com vários protocolos não são necessariamente bons testes de segurança. Por definição, testes de conformidade são positivos, ou seja, não testam entradas com falhas. Mesmo com os testes de conformidade, não há garantias de que tais ferramentas estejam protegidas de ataques.

9.3.2 Análise Estática

Anaĺise estática é um método onde o código fonte é analisado para encontrar problemas de segurança em potencial. Ferramentas de análise são diferentes na maneira em que operam, mas uma tipicamente detectam a utilização de funções ou chamadas ao sistema que são fontes de erro em potencial, ou então determinam o fluxo dos dados através dos componentes.

Análise estática pode produzir uma grande quantidade de falso positivos, e normalmente é necessário identificar alguns desses dados para ignorá-los. Algumas ferramentas mais simples talvez não conseguem diferenciar entre chamadas seguras ou não a funções potencialmente danosas para o sistema, enquanto que as mais sofisticadas testam várias possíveis

entradas da funcção para avaliar o resultado baseado nos dados inseridos.

A melhor utilização para análise estática é o auxílio na revisão de código. Problemas que são identificados pelas ferramentas de análise estática podem ser tratados como falha e o esforço na utilização de análise de código pode ser usado para determinar outros tipos de problemas. Um número alto de problemas apontado por uma ferramenta dessa natureza serve como indicador de que uma determinada parte do código deve ser revista manualmente.

Se o número de falsos positivos decair com sucesso, o sistema de análise estática também pode ser utilizado como um "guardião" ao utilizar ou modificar um código do sistema de versão de software ou então como uma das fases do teste de unidade. O primeiro passo para a análise estática deve ser acionar todos os possíveis avisos do compilador e listá-los sempre que possível.

9.3.3 Conflitos de Interesse

Usabilidade e Segurança

Em algumas vezes, usabilidade se torna um obstáculo para a segurança, pois tais funcionalidades geralmente impedem o livre uso do sistema. Embora isso seja uma verdade em alguns cenários (por exemplo, o conceito de certificados não é algo simples para o usuário final), é possível argumentar que esse problema reside no baixo grau de usabilidade das funcionalidades relacionadas à segurança do que o próprio conceito de segurança. Por exemplo, certificados poderiam ser, na maioria das vezes, totalmente invisíveis par ao usuário. Além disso, há novas soluções [66] propõe soluções que trazem mudanças na percepções de funcionalidades de segurança.

É importante entender que dispositivos móveis são utilizados por pessoas que não interagem com computadores (laptop, desktop, etc.) todos os dias. Espera-se que esta tendência será cada vez mais significante no futuro. Além disso, deve-se considerar que grande parte da população tem o dispositivo móvel como principal meio eletrônico de comunicação (e não o desktop) e essa parcela não possui necessariamente o mesmo modelo de segurança de um browser em suas mentes como aqueles que utilizam um browser no desktop, por exemplo.

Quando os dispositivos móveis foram criados, funcionalidades de segurança não eram levadas em consideração. Muitos dos aspectos eram relacionados à interface com usuário

bastante restrita. Como exemplo, um campo de captura de senha possui letras em maiúsculas e minúscular e provavelmente alguns caracteres. Digitar as senhas em um teclado comum de um dispositivo móvel consome mais tempo e é bastante suscetível a erros. Dessa maneira, um sistema que facilitasse essa troca de informações (ou até mesmo, utilizasse outros mecanismos de autenticação) seria bem aceito pelos usuários.

Interoperabilidade e Segurança

Interoperabilidade e segurança muitas vezes podem ser vistas em conflito. Para se obter máxima interoperabilidade um sistema "deve ser liberal em relação ao que aceita e conservador no que envia". Entretanto, o liberalismo também trás muita responsabilidade: mesmo que um sistema seja liberal em relação às entradas, ele deve ter conhecimento da origem da informação antes de tratá-la.

9.4 Conclusão

O número crescente de usuários de dispositivos móveis e a constante busca por falhas de segurança de sistemas levam à busca por aplicações mais seguras, capazes de executarem em vários ambientes, inclusive os mais hostis. Ao se desenvolver projetos *open source* para dispositivos móveis, também é importante observar as características inerentes a esse tipo de software e quais os impactos no desenvolvimento. Por fim, a fase de testes, embora não consiga cobrir a totalidade dos casos de ataque, trás um nível de qualidade aceitável, e também a implementação de funcionalidades de segurança.

Capítulo 10

Certificação de Qualidade

Programadores da plataforma maemo precisam levar em consideração muitos detalhes ao desenvolver aplicações de qualidade para dispositivos embarcados. Em comparação com os sistemas desktop, no desenvolvimento de aplicações para dispositivos embarcados o desenvolvedor precisa se preocupar com desempenho, utilização de recursos, consumo de energia, segurança e questões sobre usabilidade. Neste capítulo são discutidos aspectos relativos à qualidade de uma aplicação maemo, ou seja, quais os aspectos que o desenvolvedor precisa considerar ao criar aplicativos diversos para a plataforma maemo.

10.1 Introdução

O usuário de um Internet Tablet tem a nacessidade de modificar o conjunto de aplicativos existentes na plataforma ao adicionar ou remover alguma aplicação. Dessa forma, é importante prover um mecanismo simples e eficiente para gerência das aplicações presentes na plataforma, e que ofereça baixo grau de dificuldade ao usuário.

A plataforma maemo é baseada em uma distribuição Linux/Debian. Ela possui uma maneira elegante para gerenciar os aplicativos presentes no sistema. No maemo, as aplicações são distribuídas através de pacotes .deb (pacotes Debian), os quais podem ser instalados ou removidos da plataforma. Os pacotes estão concentrados em repositórios que podem ser hospedados em qualquer lugar. Porém, grande parte das aplicações estão presentes em repositórios oficiais da plataforma maemo. É possível realizar tais tarefas através da linha de comando (usando as ferramentas dpkg) ou uma interface gráfica conforme ilustrado na

Figura 10.1.

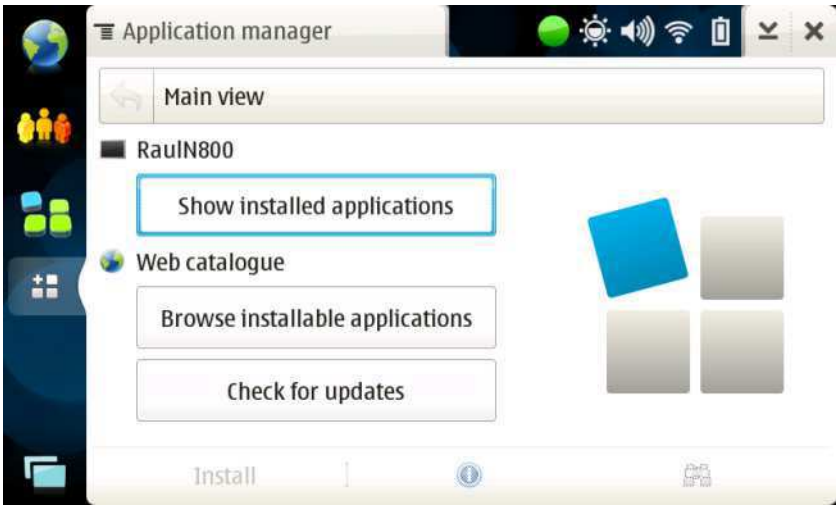

Figura 10.1: Gerenciador de pacotes da plataforma maemo.

Maemo é uma plataforma aberta e não possui uma entidade certificadora de qualidade para aplicações maemo (disponíveis em repositórios), assim como existe para Symbian [1]. Com exceção as próprias aplicações (pacotes) da Nokia, todas as demais não são "certificadas". Dessa forma, foram sugeridas algumas atividades a serem realizadas durante o processo de desenvolvimento que aumentam as chances de se obter uma aplicação maemo com qualidade. Nas próximas seções também serão discutidos aspectos que devem ser considerados ao desenvolver aplicações para a plataforma maemo.

10.2 Desempenho e Tempo de Resposta

O usuário deve ter o controle de todo o dispositivo, o qual deve dar respostas a todas as ações durante todo o tempo. A interface gráfica pode dar respostas ao usuário das seguintes maneiras: i) usando eventos para exibir um diálogo que informa o estado e bloqueando a interface gráfica enquanto estiver operando; ii) dividindo a operação em partes e exibir o progresso, por exemplo, barra de progressos e iii) utilizando de threads com o auxílio da biblioteca GLib, sendo esta mais difícil de implementar e depurar. Para tarefas que consomem mais tempo, a segunda opção é recomendada. Se a operação é realiza em poucos segundos,

[1] `https://www.symbiansigned.com/app/page/EndUserStatement`

a primeira opção deve ser utilizada.

Assim como em outras plataformas, o excesso de utilização da CPU deve ser evitada (por exemplo, operações redundantes, como atualizações desnecessárias da tela). Além disso, quando uma aplicação não está visível, ela deve ficar inativa para que a aplicação que está sendo atualmente utilizada pelo usuário não tenha o desempenho afetado. A plataforma disponiliza algumas ferramentas para monitorar as respostas e desempenho das aplicações [2].

10.3 Utilização de Recursos

Os recursos devem ser cuidadosamente utilizados pelas aplicações maemo, pois se uma aplicação consumir muitos recursos da plataforma (memória RAM e *flash*, energia, interfaces de comunicação, descritores de arquivos, etc.) ela não pode ser executada. Além da gerência correta dos recursos, problemas de desempenho e falhas (vazamento de memória, por exemplo) devem ser evitados. Tal tarefa pode ser auxiliada pelas ferramentas existentes no maemo SDK, tais como Valgrind.

10.4 Consumo de Energia

O dispositivo torna-se mal utilizado quando o usuário deve recarregá-lo todo o tempo ou quando há o consumo excessivo enquanto estiver desocupado. Dessa forma, laços de espera devem ser evitados. Uma boa solução deve ser baseada em eventos, por exemplo, o laço principal GTK (invocado através do comando `gtk_main()`). Recomenda-se também evitar o uso freqüente de temporizadores. No gráfico ilustrado na Figura 10.2 estão ilustradas as principais atividades que mais consomem energia em dispositivos móveis.

10.5 Testes de Qualidade

Embora sejam utilizadas várias ferramentas que auxiliam a detecção de falhas, existem alguns problemas que só podem ser encontrados caso um cenário de utilização seja executado e encontrem-se os problemas esperados. Por exemplo, não é possível determinar, através de

[2] `http://maemo.org/development/tools/`

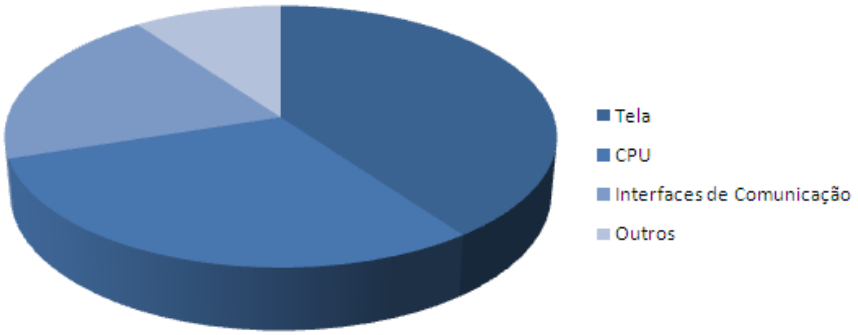

Figura 10.2: Consumo de energia em dispositivos móveis.

ferramentas, se um pacote está sendo devidamente instalado. A seguir, descrevem-se alguns conceitos que devem ser considerados no momento dos testes das aplicações.

Empacotamento Ao disponibilizar uma aplicação através de pacotes Debian, é necessário verificar como esse pacote se comporta no sistema de gerenciamento. Assim, o desenvolvedor deve observar como se comporta a aplicação e o gerenciador de pacotes ao instalar, atualizar ou remover o pacote Debian. Para verificar se houve erros durante algum desses processos, o desenvolvedor pode consultar o log do gerenciador de aplicativos;

Conectividade As interfaces de comunicação (Wi-Fi, Bluetooth e WiMax) podem ser desativadas quando o dispositivo muda de estado (para o estado *offline*, por exemplo) ou mesmo simplesmente não há conexão disponível. Dessa forma, é importante testar se a aplicação está pronta para cenários onde há ausência de conexão;

Cartão de memória O cartão de memória é uma importante memória secundária do Internet Tablet. Mesmo se a aplicação não lê informações do cartão de memória, é importante realizar alguns testes, como por exemplo: verificar o comportamento do software ao se remover e inserir um cartão no dispositivo, bem como testar o que ocorre quando não há cartão de memória disponível ou o mesmo não possui mais espaço;

Câmera No modelo N800, a câmera é retráctil e pode ser utilizada a qualquer momento.

Caso a aplicação utilize as funcionalidades da câmera, é necessário verificar o seu comportamento quando a câmera está ou não sendo utilizada e também testar se o uso da câmera por outras aplicações afeta aquela em teste;

Integração com a Plataforma Para verificar se a aplicação está devidamente integrada, deve-se observar aspectos de interface gráfica e armazenamento de dados. Em relação à interface gráfica, o aplicativo deve apresentar todos os elementos gráficos (botões, caixas de texto, etc.) e assegurar que estes estão com o tema Hildon. Outro aspecto de interface gráfica é a integração com o método de entrada: sempre que um elemento de entrada de texto (caixa ou campo de texto) ganha o foco, o teclado virtual ou o reconhecedor de escrita deve aparecer na tela, diminuindo o espaço visível da aplicação. É importante testar se o backup do sistema está salvando corretamente as configurações do aplicativo que está sendo testado;

Desempenho A aplicação dever ser inicializada e finalizada de maneira rápida. O ideal é que o tempo de inicialização seja menor do que 2 segundos na primeira vez e menor do que 1 segundo nas demais. Para a finalização, a aplicação não pode ultrapassar 2 segundos;

Utilização de Bateria Conforme visto anteriormente, uma das atividades que mais consomem energia é o processamento. Dessa forma, é necessário verificar se a aplicação utiliza a CPU de maneira correta, por exemplo: quando o Internet Tablet estiver em estado de espera, não pode haver nenhuma atividade de processamento da aplicação;

Confiança Alguns erros não são determinados nos primeiros usos do software, sendo necessário cenários bastante complexos para serem reproduzidos. Vazamento de memória e dead lock são exemplos desses tipos de erros. O desenvolvedor tem a sua disposição ferramentas que realizam os testes de maneira automática (Valgrind, por exemplo);

Escalabilidade Para aplicações que obtêm dados de uma fonte externa (do sistema de arquivos ou da própria rede), deve-se testar qual o comportamento ao se utilizar dados de tamanhos diferentes (por exemplo, dados de 1KB, 10KB, 100KB, 1MB ou 10MB).

Também é importante verificar como a aplicação se comporta em diferentes tamanhos de tela (visão normal ou tela cheia);

Documentação Alguns artefatos de documentação devem ser disponibilizados, tanto para consulta dos usuários quanto para os próprios desenvolvedores: release note, changelog, documentações de API e cópia da licença de distribuição do software;

Conformidade com Licensas O desenvolvedor deve verificar questões relativas a direitos de cópia e licenças da aplicação. É importante também observar se a licença determina que o código fonte deva ser disponibilizado.

10.6 Conclusões

Embora não existam meios oficiais para atestar a qualidade de uma aplicação maemo, muitos foram os esforços da comunidade para elaborar metodologias que, caso utilizadas no desenvolvimento, aumentam as chances de se obter um software com um alto grau de qualidade. Para aplicações embarcadas, não é necessário apenas obter um software livre de falhas. Além disso, é importante utilizar corretamente os recursos disponíveis (memória, processamento, interfaces de rede, tela, etc.) e seguir uma série de padrões estabelecidos para a plataforma, com o intuito de gerar um aplicativo de qualidade. Dessa forma, cenários de testes auxiliam bastante na tarefa, pois definem bem quais os aspectos a serem cuidadosamente analisados (tema Hildon, documentação, escalabilidade, etc.).

Capítulo 11

Ferramentas de Desenvolvimento

Assim como outros profissionais que desenvolvem aplicações para desktop, os desenvolvedores de software embarcado também precisam de compiladores, interpretadores, linkers, ambientes integrados de desenvolvimento e outras ferramentas utilizadas no processo de produção de software. As ferramentas para software embarcado são diferentes, pois executam em uma plataforma enquanto produzem software (executáveis) para uma outra plataforma. Por isso tais ferramentas também são conhecidas como ferramentas de desenvolvimento *cross-platform*, ou ferramentas de *cross-development*. Neste capítulo, discute-se a configuração e o uso de ferramentas de desenvolvimento *cross-platform*.

11.1 Desenvolvimento de Aplicações Embarcadas

Desenvolvimento de software para sistemas embarcados é diferente dos demais: o ambiente no qual a aplicação é desenvolvida difere do ambiente no qual ela é executada. Compiladores, interpretadores, linkers e outras ferramentas devem produzir artefatos que possam ser executados perfeitamente na plataforma alvo [92]. É importante a existência de ferramentas que auxiliam também a implantação de aplicações, ou seja, permitam transferir e executar o aplicativo diretamente no Internet Tablet. Geralmente, utilizam-se ferramentas de rede (`ssh`, `sftp`) bastante conhecidas. Contudo, em algumas situações, faz-se necessária a depuração e até mesmo gerar dados referentes ao desempenho da aplicação (perfilação). A plataforma maemo também possui ferramentas que auxiliam a análise da aplicação, com o intuito de auxiliar o programador a encontrar e corrigir bugs, bem como melhorar também o

desempenho do software.

11.1.1 GNU *Autotools*

O projeto GNU oferece um conjunto de ferramentas bastante úteis para o desenvolvimento de aplicações pora Linux embarcado. Trata-se das ferramentas *Autotools*, as quais auxiliam na geração de software (executável e bibliotecas) portáveis para diversos sistemas baseados em Unix. As principais ferramentas são `autoconf` e `automake`.

O `autoconf` é uma ferramenta para produção de *scripts* que verificam as dependências necessárias (bibliotecas, versões, compiladores, etc.) da plataforma para a correta construção do software. Uma vez gerados, esses *scripts* são independentes, podendo ser executados sem a necessidade do `autoconf`. A ferramenta processa alguns arquivos de entrada (`configure.in` ou `configure.ac`, mas `configure.ac` é geralmente utilizado) para gerar um *script* `configure`.

O `automake` é utilizado para facilitar a geração de arquivos `makefile`, os quais automatizam o processo de compilação e até mesmo a implantação da aplicação. O desenvolvedor não precisa se preocupar em editar várias targets do arquivo `makefile`, pois elas são geradas pela própria ferramenta `automake` a partir de arquivos de entrada (`makefile.in` ou `makefile.ac`). Tais arquivos podem ser gerados pelo *script* configure, produzido pela ferramenta `autoconf`.

11.1.2 Ambiente de Desenvolvimento

Cross-development é uma técnica relativamente nova na área de desenvolvimento de software. Surgiu da necessidade de se desenvolver uma aplicação para uma arquitetura de hardware diferente daquela utilizada como ambiente de desenvolvimento, por exemplo, desenvolver um software embarcado em um desktop (arquitetura x86) para ser executado em um Internet Tablet (arquitetura baseada em ARM).

Plataformas embarcadas possuem uma série de limitações, inclusive poder de processamento. Portanto, a compilação de uma aplicação embarcada, por exemplo, não pode ser realizada na plataforma alvo, uma vez que é necessário um poder de processamento maior e também memória: ambos escassos na grande maioria das arquituras embarcadas. Assim,

gerar o executável em uma outra máquina com maior disponibilidade de recursos diminui o tempo necessário.

A idéia básica do método de *cross-development* é utilizar algum processador (HOST) mais poderoso com o intuito de desenvolver o software para um outro processador (TARGET) com algumas limitações. Dessa forma, a máquina que compila a aplicação não pode executar nativamente o software compilado. O principal problema dessa abordagem está em garantir a geração de código compatível para a arquitetura TARGET.

Existem dois tipos de ambientes para *cross-development*: *prontos para uso*, nos quais o programador precisa apenas modificar certas variáveis; e *construção personalizada*, que exigem tempo e esforço extra do desenvolvedor para uma compilação completa [92].

Na construção personalizada, é possível utilizar as ferramentas GNU: binutils, gcc, glibc e alguns pacotes (patches) específicos. Contudo, configurar e compilar um ambiente através dessa abordagem é uma operação complexa e delicada que requer um bom entendimento das dependências entre os diversos pacotes de software e suas respectivas regras. Esse conhecimento é necessário, pois os componentes utilizados são desenvolvidos e disponibilizados de maneira independente.

É necessário, para a construção do ambiente, selecionar as versões dos componentes que serão utilizados. Tal tarefa envolve a escolha da versão das ferramentas binutils, gcc e glibc. Uma vez que tais componentes são independentes, não se garante a correta construção do ambiente de desenvolvimento. Pode-se até mesmo tentar utilizar as versões mais recentes. Contudo, também não há garantias de seu funcionamento. Existem tabelas que especificam quais são as versões corretas de cada uma das ferramentas a serem utilizadas. Com as ferramentas nas versões corretas, resta apenas construir o ambiente. O processo de compilação, instalação e configuração das ferramentas também é complexo e demorado.

O maior problema presente nesta abordagem de construção personalizada reside na ausência de flexibilidade. Por exemplo, caso o programador deseje utilizar uma outra plataforma (MIPS), é necessário refazer todo o ambiente, escolher novamente as versões corretas e compilá-las. Até mesmo a mudança de uma ferramenta específica presente no ambiente é complicada.

Uma solução bem mais prática são os ambientes de *cross-development* prontos para uso. Tais ambientes são pré-configurados e facilitam bastante a criação de um ambiente de de-

senvolvimento. A plataforma maemo adota uma solução pronta, o Scratchbox, o qual será descrito na próxima seção.

Scratchbox

Desenvolvido pela empresa Movial como um projeto *open source*, Scratchbox é um ambiente de compilação e configuração para a construção de software e distribuições Linux para algumas plataformas: ARM,X86,MIPS, dentre outras. Ele oferece ao desenvolvedor um ambiente que possui todas as características da plataforma alvo. A solução proposta pelo Scratchbox está ilustrada na Figura 11.1.

O ambiente possui diversas ferramentas para *cross-development* de software para várias plataformas, como ARM e MIPS. O desenvolvedor precisa definir uma TARGET no ambiente antes de iniciar o desenvolvimento, isso porque todas as ferramentas são configuradas automaticamente de acordo com a TARGET escolhida [59]. A TARGET define a plataforma alvo (ARM, Sparc, etc.), a biblioteca C (glibc, uclibc), o modo de emulação (qemu-arm,qemu-ppc,etc.) e os devkits - conjuntos de ferramentas particulares. Todo o ambiente utilizado no processo de desenvolvimento está presente em um ambiente igual à plataforma TARGET. Essa abordagem é chamada de *sandboxing* (sandbox = caixa de areia). A criação de TARGET tem o auxílio de uma interface gráfica.

O objetivo principal do Scratchbox é oferecer aos desenvolvedores, os quais precisam ter acesso ao dispositivo para testar suas aplicações, um ambiente que possui todas as características da plataforma TARGET. Muitas vezes é necessário que este ambiente esteja disponível antes mesmo do dispositivo estar pronto. O Scratchbox é um ambiente desenvolvido para facilitar o desenvolvimento de aplicações embarcadas para Linux, além de prover um conjunto amplo de ferramentas para integrar e também compilar uma distribuição Linux completa para a plataforma TARGET.

Todo o ambiente é devidamente configurado para a plataforma alvo escolhida: as variáveis de ambiente são modificadas, e links simbólicos são criados para que o programador utilize os mesmos comandos normalmente executados no processo de desenvolvimento desktop. Por exemplo, para invocar o compilador que deve gerar o código para a arquitetura ARM (`arm-linux-gcc`), o programador utiliza o mesmo comando do ambiente desktop: `gcc`. O Scratchbox cria um link simbólico entre o comando **gcc** e o compilador real

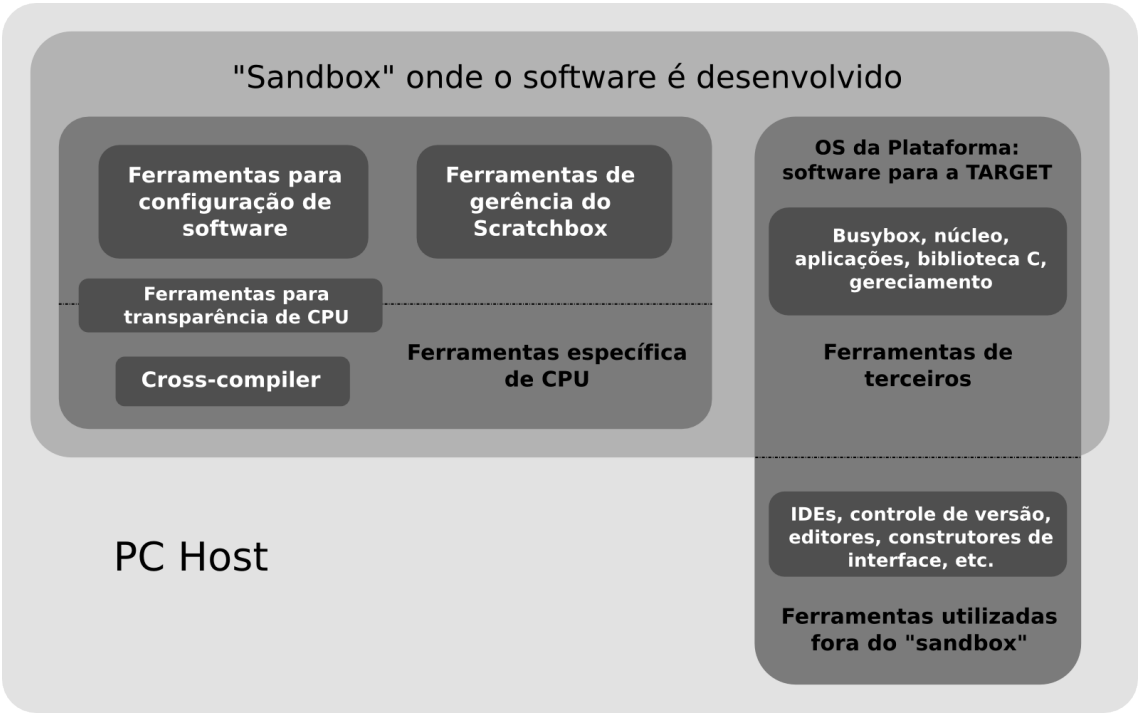

Figura 11.1: Funcionamento do Ambiente Scratchbox.

(arm-linux-gcc). O código é compilado utilizando comandos GCC [36] padrão como se o desenvolvimento fosse realizado no próprio ambiente desktop. Várias tarefas do desenvolvimento, tais como projeto, codificação e teste podem ser realizadas no Scratchbox mesmo sem a plataforma alvo estar disponível. A inexistência da plataforma alvo não afeta o ciclo de desenvolvimento.

11.1.3 Depuração

A depuração é um processo bastante útil para detectar falhas em um software. O programador tem acesso a várias informações do programa (variáveis, símbolos) e da própria plataforma (registradores) durante a depuração do aplicativo. Além disso, é possível acompanhar a execução de cada linha do programa. Para depurar aplicações embarcadas para a plataforma Linux, utiliza-se a ferramenta GDB.

gdb

O depurador GNU (GNU debugger - gdb) é um depurador simbólico do projeto GNU, sendo uma das ferramentas mais importantes e utilizadas para depuração em qualquer sistema Linux. Criado e mantido há mais de 10 anos, é utilizado até mesmo por sistemas embarcados que não utilizam o sistema operacional Linux, permitindo a depuração da aplicação remotamente. Além de permitir a depuração de aplicativos localmente, também é possível depurar aplicações que estão em outros ambientes, inclusive no próprio Internet Tablet. Isso é possível graças ao `gdbserver`, um aplicativo que é executado remotamente e se conecta à instância do `gdb` na máquina local, possibilitando a depuração. A conexão pode ser realizada através de uma rede TCP/IP ou através de um cabo serial. A possibilidade de depurar a aplicação remotamente, sem a necessidade de instalar o GDB no ambiente remoto, é muito importante, pois economiza memória (o tamanho do binário `gdb` é grande para um sistema embarcado e possui cerca de 2.5Mb).

GDB oferece muitas facilidades para acompanhar e alterar a execução de programas. O programador pode monitorar e modificar os valores das variáveis internas do programa e mesmo modificar o fluxo de controle normal do aplicativo. O GDB possui versões para as seguintes plataformas: Alpha, ARM, AVR, H8/300, System/370, System 390, X86 e X86-64, IA-64 "Itanium", Motorola 68000, MIPS, PA-RISC, PowerPC, SuperH, SPARC e VAX. O projeto está em pleno desenvolvimento. Uma das principais funcionalidades a serem adicionadas é o suporte à "depuração reversível". Tal funcionalidade permite que uma sessão de depuração possa voltar no fluxo de execução para rever de maneira mais detalhada o que aconteceu.

A principal limitação do GDB é a ausência de interface gráfica para o usuário (sua interface padrão é linha de comando). Vários front-ends para o GDB já foram criados, por exemplo DDD, Eclipse CDT, Xcode e GDBtk/Insight. Essas ferramentas oferecem facilidades similares aos depuradores encontrados em ambientes integrados de desenvolvimento.

11.1.4 Perfilamento

A técnica de perfilamento (*profiling*) consiste em uma análise de desempenho para determinar o comportamento de um programa durante sua execução. O resultado pode ser um fluxo

de dados obtido a partir da execução do programa ou um conjunto de dados estatísticos dos eventos observados. Perfiladores usam uma variedade de técnicas para coletar dados, incluindo interrupções de hardware, instrumentação de código e contadores de desempenhos.

No processo de desenvolvimento de sistemas embarcados, é importante utilizar perfiladores para determinar possíveis problemas relacionados ao mau uso dos recursos da plataforma, por exemplo, memória e processamento. No desenvolvimento de aplicações maemo, pode-se utilizar duas ferramentas de perfilamento: Valgrind e OProfile.

Valgrind

Valgrind [68] é uma suíte de ferramentas para depuração e perfilação de programas que executam na plataforma Linux (x86, amd64, ppc32 e ppc64). O sistema consiste de uma parte central (core), que provê uma CPU simulada (ou seja, implementada via software). Também possui um conjunto de ferramentas utilizadas para realizar tarefas de depuração, perfilação e atividades similares. A arquitetura é modular, de maneira que novas ferramentas podem ser adicionadas sem afetar o funcionamento do restante da estrutura. Valgrind insere código para capturar informações importantes para análise posterior, o que torna o programa de 10 a 300 vezes mais lento dependendo da ferramenta utilizada. Contudo, é o perfilador que auxilia bastante na depuração do programa.

As principais ferramentas são as seguintes:

Memcheck Ferramenta para detectar problemas relacionados ao gerenciamento de memória nas aplicações. Todas as leituras e escritas na memória são verificadas e as chamadas às funções do sistema malloc/new/free/delete são interceptadas. Dessa forma, `memcheck` pode encontrar os seguintes problemas:

- Uso de memória não inicializada;
- Escrita/Leitura de memória após ser liberada;
- Escrita/Leitura ao final de um bloco alocado;
- Escrita/Leitura em áreas impróprias da pilha;
- Vazamento de memória, ou seja, quando os ponteiros para os blocos de memória são perdidos;

- Uso incorreto de *malloc/new* versus *free/delete*;
- Sobrescrita de ponteiros destinos e origem em funções como `memcpy()`.

Problemas como esses são difíceis de serem diagnosticados. Caso permaneçam escondidos, podem ocasionar problemas sérios na aplicação difíceis de serem encontrados.

Cachegrind É um perfilador para memória cache. Realiza simulação detalhada nos caches I1, D1 e L2 da CPU que possibilitam determinar problemas relativos a esse tipo de memória. A ferramenta disponibiliza a quantidade de falhas de cache, referências de memória e quantidade de instruções em cada linha de código. Também gera sumários baseados em função, módulos e todo o programa. Nas arquiteturas x86 e X86-64, o Cachegrind detecta qual a configuração de cache da máquina usando a instrução CPUID. Dessa forma, não é necessário nenhuma configuração extra;

Massif Ferramenta que realiza perfilação do heap. Calcula quanto de memória *heap* está sendo utilizada pelo programa. É possível a disponilização de informações como quantidade de blocos na memória *heap* e tamanhos de pilhas. A perfilação da memória heap pode auxiliar na diminuição de memória utilizada pelo programa. Em máquinas mais novas com memória virtual, a boa utilização da memória *heap* pode reduzir as chances do programa estourar a capacidade de memória existente, tornando a aplicação mais rápida através da redução da quantidade de paginação necessária;

Helgrind Detecta erros de sincronização em programas que utilizam as primitivas de threads POSIX (pthreads). É possível detectar as seguintes classes de erros: i) má utilização da API POSIX pthreads, ii) trechos de código com grande chances de ocorrer *deadlock* ou iii) *data race* (acesso à memória sem o bloqueio correto). Problemas como esses são praticamente impossíveis de serem reproduzidos outras vezes. Assim, *deadlocks* e outras falhas tornam-se bastante difíceis de serem identificados sem o auxílio de ferramentas apropriadas.

Existem outras ferramentas(Lackey, Nulgrind e Callgrind) que também estão disponíveis. Algumas (Lackey, Nulgrind) não possuem funcionalidades úteis: são apenas utilizadas para ilustrar como é possível criar ferramentas simples e auxiliar os desenvolvedores do Valgrind

em várias maneiras. A ferramenta Nulgrind, por exemplo, não adiciona nenhuma instrumentação no código. Lackey é uma ferramenta simples que conta quantas instruções, acesso à memória e número de operações com inteiros e ponto flutuante são realizadas pelo programa.

Valgrind está bastante relacionado com detalhes da CPU e do sistema operacional, e em menor grau com o compilador e com bibliotecas básicas do sistema. A versão 3.3.0 suporta as seguintes plataformas: x86/Linux (estável), amd64/Linux (estável), ppc32/Linux e ppc64/Linux (menos estável, mas funciona bem). Também há suporte experimental para ppc32/AIX5 e ppc64/AIX5. Valgrind não está disponível para a arquitetura ARM. Dessa forma, a utilização da ferramenta no processo de desenvolvimento de aplicações maemo ocorre dentro do ambiente X86 no Scratchbox.

OProfile

OProfile [3] é um perfilador baseado em estatísticas. Ou seja, os resultados são gerados através da coleta regular do estado dos registradores de cada CPU e convertidos em dados úteis para o programador. A ferramenta OProfile obtém esses dados a partir do fluxo de amostras de valores do registrador PC, como também os detalhes de qual tarefa estava sendo executada no momento da interrupção. As aplicações utilizam a chamada de sistema `mmap` para estabeler um mapeamento entre o endereço de espaço de um processo e um arquivo ou objeto presente na memória. Dessa forma, é possível encontrar o arquivo binário e o *offset* a partir da leitura das listas das tarefas presentes nas áreas de memória mapeadas. Cada valor do PC é então convertido em uma tupla no formato [imagem-do-binário,offset]. Essa abordagem também é utilizada por outras ferramentas para determinar qual a origem do código, incluindo instruções assembly, símbolos e código fonte (através de informações de depuração presente no código).

A leitura regular do valor do PC, conforme descrito anteriormente, gera um resultado bastante satisfatório. É possível determinar o que foi realmente executado e com que freqüência, obtendo dados estatísticos bem próximos da execução do software. Em operações comuns, o tempo entre cada interrupção para coleta de dados é definido através de um número fixo de ciclos de relógio. Isto significa que os resultados refletem em quais momentos a CPU está utilizando mais tempo de processamento, informação esta bastante útil para análise de desempenho.

Contudo, em algumas situações, o programador precisa de tipos diferentes de informação de desempenho: por exemplo, o programador precisa saber em quais rotinas há a maior ocorrência de falhas de cache. Dessa forma, com o aumento na importância em tipos de métricas como estas relatadas levou a vários fabricantes de CPU a proverem, a nível de hardware, contadores de desempenho capazes de obter os eventos de hardware. Geralmente, tais contadores são incrementados a cada evento recebido e uma interrupção é gerada a cada X números de eventos recebidos. OProfile utiliza essas interrupções para geração de estatísticas. Dessa forma, os resultados de perfilação são uma aproximação estatística de quanto eventos um determinado trecho de código gerou.

Considere um software simples que executa somente duas funções: A e B. A função A gasta um ciclo para ser executado, enquanto a função B gasta 99 ciclos. A CPU utilizada processa a uma velocidade de 100 ciclos por segundo e o contador de desempenho foi configurado para que seja gerada uma interrupção a cada X números de eventos (neste caso, cada evento é um ciclo de relógio). As chances de ocorrer uma interrepção na função A é de 1/100 e na funcção B é de 99/100. Assim, obtemos uma aproximação estatística dos dados de desempenho de cada uma das funções ao longo do tempo. A mesma análise pode ser realizada outros tipos de eventos, considerando que uma interrupção é gerada a cada N eventos.

Normalmente, existem mais de um desses contadores de desempenho. Dessa forma, é possível modificar as configurações de perfilação para capturar diferentes tipos de eventos. Ao utilizarmos esses contadores, obtemos ganho em qualidade nas métricas de desempenho, assim como uma sobrecarga menor para o sistema. Se o OProfile, ou então a CPU, não suportar contadores de desempenho, então um método mais simples é utilizado: as interrupções de tempo do núcleo são utilizadas para gerar as métricas.

11.1.5 Problemas e Limitações

Conforme descrito anteriormente, existem diversas ferramentas que auxiliam no desenvolvimento de aplicações para a plataforma maemo, por exemplo editores de texto (`vi`, `nano`, `gedit`), compiladores (`gcc`), interpretadores e depuradores (`gdb`). Ocasionalmente, as ferramentas utilizadas no processo de desenvolvimento de software são independentes e cada uma apresenta uma interface própria com atalhos e layout gráfico com características

próprias. Um dos fatores que dificulta o desenvolvimento de novos aplicativos é a inexistência de ferramentas integradas para as várias fases de um processo desenvolvimento de software, principalmente no caso dos dispositivos baseados no Linux Embarcado.

No caso do desenvolvimento para o Internet Tablet, utilizam-se ambientes independentes para um processo de desenvolvimento. Conseqüentemente, a produtividade do desenvolvedor fica limitada pela mudança constante de contexto para executar tais etapas. Deste modo, é relevante e atual conceber ambientes integrados que simplifiquem as etapas do processo de desenvolvimento para aumentar a produtividade dos programadores e acelerar a oferta de novos aplicativos.

11.2 Conclusões

Existem ferramentas bastante úteis que facilitam o desenvolvimento de aplicações Linux embarcadas: `gcc`, `ld`, `vi`, `vim` , `doxygen`, etc. Tais ferramentas de desenvolvimento para a plataforma maemo (Linux embarcado) são um pouco diferentes daquelas utilizadas no ambiente desktop: os arquivos executáveis são gerados para uma outra arquitetura alvo, ou seja, para um processador distinto daquele usado no desenvolvimento. O principal problema existente é a ausência de um ambiente único de desenvolvimento, o qual facilita o uso de várias ferramentas ao mesmo tempo. Um ambiente de desenvolvimento integrado procura reunir o maior número possível de ferramentas sob uma mesma interface, com o intuito de suportar todo o processo de criação de software. Dessa forma, o programador não gasta muito tempo e esforço para gerenciar a utilização de várias ferramentas. No próximo capítulo (Capítulo 12, exploram-se as vantagens trazidas pelos ambientes integrados de desenvolvimento. Também são apresentados dois ambientes integrados criados para dar suporte ao desenvolvimento de aplicações maemo: ESbox e PluThon.

Capítulo 12

Ambientes Integrados de Desenvolvimento

Neste capítulo, são discutidos alguns problemas existentes na utilização de ferramentas para auxiliar o processo de desenvolvimento de software para dispositivos móveis. Com o intuito de automatizar as diversas tarefas existentes em um processo de desenvolvimento, a utilização de várias ferramentas independentes nem sempre traz bons resultados. Os Ambientes Integrados de Desenvolvimento (também conhecidos como *Integrated Development Environment* - IDE) provêem soluções que facilitam o trabalho do programador. Serão apresentados, também, duas ferramentas desenvolvidas no Laboratório Embedded que buscam facilitar as tarefas de desenvolvimento, melhorando a produtividade no projeto, bem como facilitando o aprendizado de novas tecnologias. Ambas as ferramentas, ESbox e PluThon, foram desenvolvidas para auxiliar a contrução de aplicações para a plataforma maemo e foram adotadas como os ambientes de desenvolvimento oficiais da plataforma.

12.1 Motivação

O processo de desenvolvimento de software é composto por diversas tarefas realizadas de maneira iterativa: análise, codificação, documentação, testes, integração, dentre outras. Geralmente, as tarefas requerem uma entrada e produzem um artefato, utilizado em fases posteriores. Com o intuito de automatizar as tarefas existentes, existem diversas ferramentas que auxiliam o membro da equipe de desenvolvimento (gerente, programador, testador, etc.)

na realização de sua atividade [25; 46].

Ocasionalmente, as ferramentas utilizadas no processo de desenvolvimento são independentes, e cada uma apresenta um ambiente gráfico próprio com atalhos, disponibilização de botões, cores e layout gráfico típicos. Contudo, por conta da independência dos ambientes, a produtividade do programador pode ser comprometida: é necessário mais tempo e esforço, considerando as mudanças constantes entre tais ferramentas [63].

Cada ferramenta possui sua própria interface, necessitando de tipos distintos de entrada e produzindo tipos de saída distintos. Assim, o desenvolvedor também deve se preocupar com o gerenciamento das ferramentas que estão sendo utilizadas. Outro fator importante é o tempo gasto para que o programador se adapte a uma nova ferramenta inserida no ambiente de trabalho. É necessário um tempo por parte do programador para treinamento e aprendizagem [63].

12.2 Ambientes Integrados de Desenvolvimento

Para solucionar o problema de integração de ferramentas em um processo de desenvolvimento de software, foram criados os IDEs. A proposta de uma IDE é reunir, em um único ambiente, grande parte das ferramentas usadas no desenvolvimento de aplicações, de forma que a utilizaccão de recursos externos ao ambiente seja minimizado ao máximo. Dessa maneira, o desenvolvedor não precisa se preocupar com o gerenciamento das ferramentas utilizadas, tampouco com as entradas exigidas e as saídas produzidas. Os IDEs procuram unificar os formatos de saída/entrada de dados da ferramentas para que o fluxo do processo ocorra dentro do ambiente.

A utilização de IDEs torna o processo de desenvolvimento mais produtivo, uma vez que as tarefas são realizadas em um só local, sem a mudança entre interfaces gráficas distintas [11]. Outro ganho está na facilidade em se adicionar uma nova ferramenta ao processo: uma vez que o desenvolvedor está familiarizado com o ambiente, o tempo utilizado para treinamento e aprendizado do novo recurso é bem menor se comparado a um ambiente de desenvolvimento descentralizado.

Existem alguns ambientes de desenvolvimento, por exemplo: Anjunta, Eclipse, Netbeans, dentre outros. Alguns destes ambientes permitem a adição de novas funcionalidades,

por exemplo, é possível adicionar um construtor de interfaces a um ambiente de desenvolvimento ou mesmo suporte a um novo compilador. A seguir, é apresentado um framework para o desenvolvimento de IDEs bastante utilizado: a plataforma Eclipse [61; 80; 7; 18; 93].

12.2.1 Plataforma Eclipse

Concebida para executar em diversos sistemas operacionais, provendo um mecanismo robusto para criação de ambientes de desenvolvimento, a plataforma Eclipse possui suporte para várias tecnologias: Java (J2SE, J2ME, J2EE), C/C++, Python, aplicações web, dentre outras. Eclipse provê um modelo de interface gráfica padrão que é utilizado pelas ferramentas adicionadas ao ambiente através de unidades chamadas *plug-ins*. A arquitetura do framework Eclipse é baseada em *plug-ins*, os quais são elementos que podem ser adicionados ou removidos da plataforma de maneira simples. Um gerente central é responsável pela correta manipulação dos *plug-ins* (adição, remoção, inicialização e finalização). Cada plug-in deve ser responsável por um pequeno número de tarefas: testes, depuração, compilação e diagramação, por exemplo.

Geralmente, uma pequena aplicação é desenvolvida como um simples *plug-in*, enquanto outras podem ser formadas por vários. Os *plug-ins* são desenvolvidos em Java, compostos por código e outros recursos, tais como imagens, arquivos de configuração, arquivos de licensa, dentre outros. Os *plug-ins* declaram pontos de extensão em arquivos. Tais pontos são necessários para que novos *plug-ins* sejam reconhecidos e adicionem suas funcionalidades na plataforma Eclipse. O modelo de interconexão é simples: um *plug-in* declara um certo número de pontos de extensão e um outro número de conexões para um ou mais pontos de extensão de outros *plug-ins*.

A arquitetura da plataforma Eclipse está ilustrada na Figura 12.1. O Eclipse disponibiliza módulos que podem ser utilizados pelas ferramentas externas adicionadas ao ambiente, por exemplo *Help*, *Team* e *Workspace*. Existem também bibliotecas de interface gráfica providas pela plataforma, tais como SWT (*Standard Widget Toolkit*) e JFace.

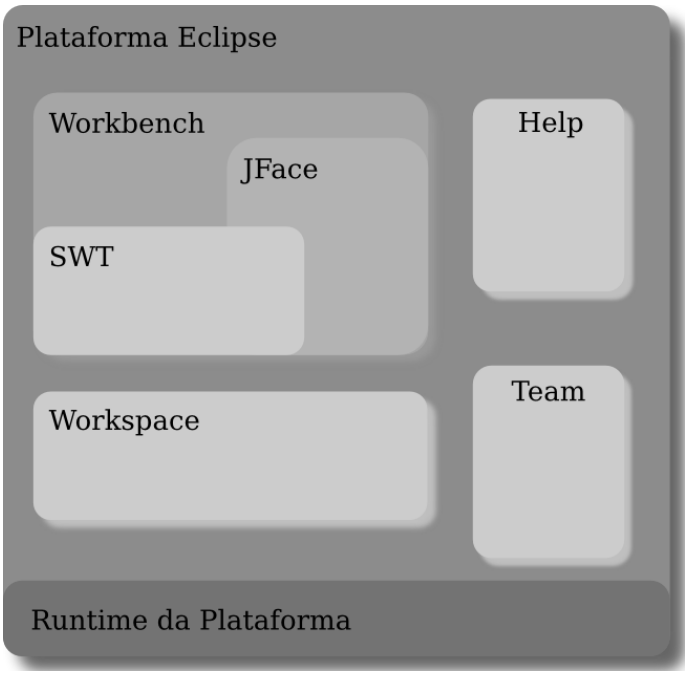

Figura 12.1: Arquitetura da plataforma Eclipse.

12.3 ESbox

Nesta seção, descreve-se a ferramenta ESbox. Desenvolvida no Laboratório Embedded, a ferramenta provê um conjunto de funcionalidades que auxiliam o desenvolvimento de aplicações maemo sobre os ambientes Scratchbox 1 e Scratchbox 2.

12.3.1 Introdução

Existe uma grande variedade de técnicas e ferramentas proprietárias e *open source* para o desenvolvimento de aplicações para Linux embarcado. Como exemplo de uma técnica bastante utilizada, podemos citar *cross-compilation* [41; 92]. Esta técnica consiste no desenvolvimento da aplicações em uma plataforma diferente daquela onde o software será executado. A plataforma de desenvolvimento, também chamada de HOST, possui poder computacional maior e mais recursos do que a plataforma alvo, chamada de TARGET.

Entretanto, para o desenvolvimento de aplicações nas quais a plataforma HOST difere da plataforma TARGET, é necessário um arcabouço de ferramentas e bibliotecas que gerem saídas próprias para o TARGET. Este arcabouço é conhecido como *cross-development*

toolchain. Existem dois tipos de *cross-development toolchain*: *prontos para uso*, nos quais o programador precisa apenas modificar certas variáveis; e *construção personalizada*, que exigem tempo e esforço extra do desenvolvedor para uma compilação completa [92].

Cada uma das ferramentas do ambiente possui sua própria interface, necessitando de diferentes formas de entrada e produzindo tipos de saída distintos. O desenvolvedor deve gerenciar corretamente o uso de tais ferramentas, demandando tempo e esforço adicionais.

A utilização dos ambientes Eclipse e Scratchbox para o desenvolvimento de aplicações para Linux embarcado pode reduzir o tempo de desenvolvimento. Entretanto, a plataforma Eclipse suporta apenas a atividade de codificação no caso de sistemas embarcados, enquanto que as atividades de compilação, depuração e execução devem ser realizadas no Scratchbox, fora do ambiente Eclipse [39].

O programador deverá mudar constantemente de ambiente para desenvolver a aplicação, usando o código produzido no Eclipse e realizando as demais atividades no Scratchbox. Esta troca de interface consome tempo e esforço do programador. Assim, verifica-se a importância de agregar as funcionalidades da plataforma Eclipse e do Scratchbox, permitindo que todo o desenvolvimento da aplicação ocorra em um único ambiente.

A solução proposta é a ferramenta ESbox, resultado da integração dos ambientes Eclipse e Scratchbox. A ferramenta foi desenvolvida focando-se o desenvolvimento sobre a plataforma maemo. É possível escrever, depurar e instalar aplicações nas linguagens C/C++ e Python compatíveis com os dispositivos que rodam essa distribuição de Linux embarcado.

No segundo semestre de 2007, o ESbox foi escolhido como ferramenta oficial para desenvolvimento de aplicações para a plataforma maemo. Os quesitos que levaram a tal escolha foram a maturidade da ferramenta, arquitetura e qualidade de código (documentação e testes de unidade). Desde então, os trabalhos foram focados em correção de bugs e adição de novas funcionalidades. O ESbox, juntamente com o PluThon e o pacote *maemo-pc-connectivity*, integram a ferramenta *Maemo Eclipse Integration*.

Atualmente, o ESbox está na versão 1.4.0 com cerca de 45 KLOC. Em janeiro/2008, uma equipe com 2 desenvolvedores e um escritor de documentação do *Nokia Research Center* em Austin/EUA juntaram-se à equipe a foram responsáveis por valiosas contribuições à ferramenta.

12.3.2 Arquitetura

A arquitetura do ESbox está ilustrada na Figura 12.2. A ferramenta executa sob o Eclipse e o toolchain Scratchbox. É composta basicamente por duas camadas: *framework* e *plug-in*. A camada *framework* comunica-se diretamente com o Scratchbox, configurando o ambiente, encapsulando comandos e verificando sua correta execução. Esta comunicação é realizada através do encapsulamento de comandos do Scratchbox em instâncias de processos na linguagem Java. Dessa forma, a ferramenta é capaz de obter da instância do processo, os fluxos de entrada, saída e erro.

A camada *plug-in* é responsável pela comunicação com a plataforma Eclipse e com os elementos da interface gráfica. Os fluxos obtidos na camada anterior são utilizados para modificar a interface com o usuário, mantendo-o informado do estado do ambiente onde sua aplicação está sendo desenvolvida. Como o ESbox é um *plug-in* da plataforma Eclipse, é possível a adição de outros *plug-ins* na ferramenta, incrementando ainda mais as suas funcionalidade.

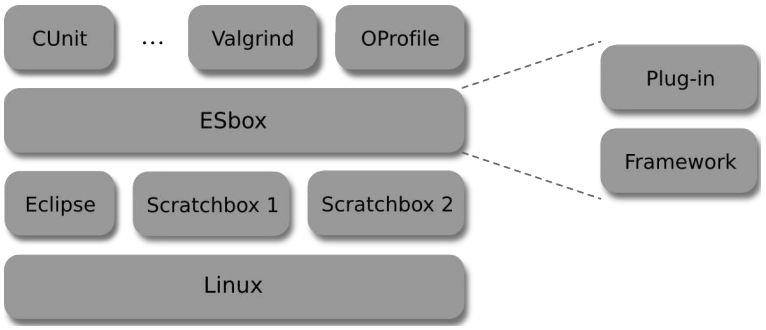

Figura 12.2: Arquitetura da ferramenta ESbox.

A ferramenta ESbox é composta por um conjunto de módulos e cada um agrega um conjunto de funcionalidades específicas, tornando o sistema mais coeso e com maior grau de independência entre os vários módulos. Na Figura 12.3 está ilustrado os módulos existentes no ESbox.

Core Módulo principal da ferramenta que implementa a camada *framework*, descrita anteriormente. Além de estabelecer comunicação com o Scratchbox, também possui elementos importantes para plug-ins Eclipse, tais como *natures*, *preference initializers* e *activators*;

UI Os principais elementos de interface gráfica estão presentes nesse módulo. Boa parte da camada *plug-in* é implementada aqui, tais como *views*, *actions* e *wizards*;

Help O framework Eclipse provê um mecanismo de documentação ao usuário integrado à plataforma, através de pontos de extensão simples de serem utilizados. Toda a documentação de ajuda ao usuário ESbox está presente nesse módulo.

Debug Módulo que implementa as funções que permitem a depuração local e remota de projetos C/C++;

Launch Implementa as funcionalidades para executar aplicações C/C++ local e remotamente;

Python UI Alguns elementos de interface gráfica mais específicos para projetos Python, tais como *wizards*, estão presentes neste módulo;

Python Debug Módulo que implementa as funções que permitem a depuração local e remota de projetos Python;

Python Launch Possui as funcionalidades para executar aplicações Python local e remotamente.

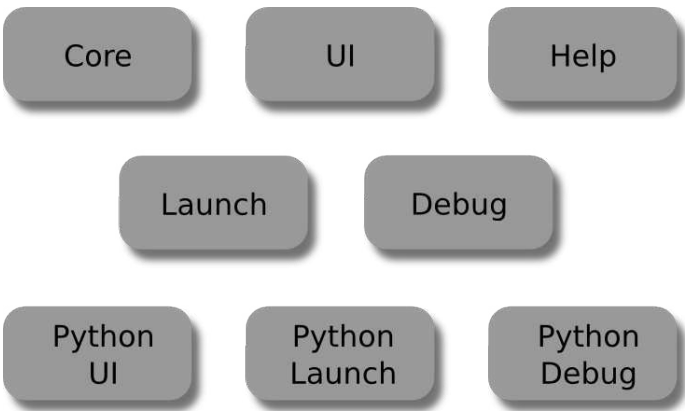

Figura 12.3: Módulos que compõem a ferramenta ESbox.

12.3.3 Requisitos

ESbox foi desenvolvida para o sistema operacional Linux, em especial a distribuição Ubuntu. A versão 1.4.0 da ferramenta requer a seguinte configuração: Scratchbox Apophis, Scratchbox 2, Eclipse 3.3.3.1, CDT 4.0.2 e PyDev 1.3.9 ou superior. Está licenciada sob EPL (Eclipse Public License) e os exemplos que podem ser utilizados pelos usuários estão licenciados sob LGPL (Lesse General Public License). Uma descrição mais completa da ferramenta, como também uma versao para instalação, pode ser encontrada em `http://esbox.garage.maemo.org`.

12.3.4 Processo de Desenvolvimento

O processo de desenvolvimento de software, quando auxiliado por IDEs, torna-se mais rápido e produtivo. Em comparação ao processo descrito no Capítulo 4, há algumas modificações que trazem ganhos para o programador. Todas as atividades descritas são realizadas em um terminal (interface textual), ou seja, desde a codificação até os testes e depuração no dispositivo. Com a adoção de ambientes gráficos para o desenvolvimento, todas as atividades envolvidas no processo podem ser realizadas dentro do Eclipse, inclusive os testes e depuração remota.

Dessa forma, observa-se ganhos tanto para o desenvolvedor de aplicativos maemo, pois há ganho de tempo e produtividade, como também para novos programadores da plataforma, já que o tempo necessário para se adaptar ao processo de desenvolvimento é menor devido ao auxílio da ferramenta ESbox.

12.3.5 Casos de Utilização

Entre o segundo semestre de 2007 e o primeiro semestre de 2008, cursos sobre a plataforma maemo foram ministrados em algumas cidades brasileiras. Durante o curso, noções de arquitetura e funcionamento da plataforma foram discutidos, bem como o processo de desenvolvimento tradicional, utilizando um ambiente com ferramentas independentes, e, na maioria das vezes, com interface textual.

Notou-se uma clara dificuldade dos alunos, principalmente, daqueles que não possuíam familiaridade com o sistema operacional Linux. Os alunos tinham que aprender os passos

necessários para a correta configuração do ambiente sempre que fosse necessário o seu uso: iniciar o Scratchbox, modificar variáveis de ambiente, iniciar o servidor X, etc. Depois de iniciar ambiente, o aluno ainda tinha outros dois problemas: codificar usando um editor de textos que não lhe era familiar (vi, vim, nano) e usar a linha de comando para compilar a executar a aplição.

A partir de maio de 2008, quando foi lançada a versão 1.4.0, mais estável, passou-se a utilizá-la como ferramenta de desenvolvimento durante os cursos. Os ganhos bastante motivadores:

Tempo e esforço menores Toda a comunicação necessária com o Scratchbox é escondida pela ferramenta, de forma que o aluno não precisa saber de quais são os comandos necessários para correta configuraçcão e utilização do ambiente de desenvolvimento maemo. O aluno inicia o desenvolvimento mais cedo, pois não há a etapa de configuração e aprendizagem do ambiente de desenvolvimento maemo. Com uma interface gráfica, fica mais fácil de se utilizar uma ferramenta.

Maior taxa de aceitação A ferramenta proporciona um ambiente gráfico com funcionalidades que facilitam as tarefas de programação, tais como codificação e depuração. O contato com o ambiente Scratchbox é praticamente inexistente da perspectiva do programador. Assim, a probabilidade de rejeitção da plataforma é menor.

Aumento de produtividade O aluno é auxiliado por um conjunto de funcionalidades importantes, tais como *code completion* e sintaxe *highlight*. Dessa forma, muitas tarefas que antes consumiam muito tempo para serem feitas, agora são realizadas de maneira fácil e rápida. Ao chegar ao final do curso, o aluno que é capaz de desenvolver aplicações sem grandes dificuldades.

É importante salientar que os vantagens obtidas com a ferramenta foram observadas durante cursos sobre a plataforma, ou seja, dentro de sala de aula. Contudo, podemos concluir que também há ganhos significativos durante o processo de desenvolvimento de aplicações.

12.4 PluThon

Nesta seção, descreve-se a ferramenta PluThon a qual auxilia o desenvolvimento de aplicações Python para a plataforma maemo. A principal diferença em relação ao ESbox reside na independência da ferramenta em relação ao ambiente Scratchbox, pois a execução é realizada diretamente no Internet Tablet.

12.4.1 Introdução

A ferramenta ESbox é bastante útil para o desenvolvimento de aplicações maemo escritas em C/C++ ou em Python. Com o uso de um ambiente de desenvolvimento tão completo quanto o ESbox, o desenvolvedor empreende menos tempo e esforço, obtendo também ganho em produtividade. Contudo, para utilizar o ESbox é necessária a instalação do ambiente Scratchbox, somente disponível em ambientes Linux. Podemos considerar tal fato como uma desvantagem, pois impossibilita o desenvolvimento de aplicações maemo em outros sistemas operacionais (Windows, MacOS).

É possível desenvolver aplicações em Python fora do ambiente Scratchbox, pois se trata de uma linguagem interpretada, sem a necessidade de compilação. Contudo, o programa deve ser executado em um Internet Tablet. Assim, o programador codifica o script Python em qualquer editor de sua escolha e deve transferi-lo para o dispositivo através de uma conexão de rede (wireless ou USB).

Cada uma das tarefas envolvidas no desenvolvimento de aplicações Python (codificação, transferência e execução) são executadas constantemente. Mais uma vez, cada tarefa é realizada em ambientes distintos, tornando o processo de desenvolvimento pouco produtivo. Assim, deve-se integrar as diferentes ferramentas envolvidas na edição, transferência de arquivos e execução remota.

Solicitou-se ao Laboratório Embedded o desenvolvimento de uma ferramenta que suprisse tais necessidades: possibilitar o desenvolvimento de aplicações Python em outros sistemas operacionais através de um ambiente integrado de desenvolvimento baseado na plataforma Eclipse. Assim, surgiu a ferramenta PluThon a qual teve o seu desenvolvimento iniciado no primeiro semestre de 2007. A ferramenta permite que o desenvolvedor crie projetos Python para maemo, edite-os usando o editor Eclipse, transfira-o para o Internet Tablet

usando a ferramenta `scp` e finalmente execute a aplicação usando `ssh`. Também é possível a depuração remota do programa.

O projeto PluThon surgiu da necessidade de desenvolver aplicações Python para a plataforma maemo sem depender do ambiente Scratchbox. Assim como o ESbox, o PluThon integra o projeto *Maemo Eclipse Integration*, desenvolvido pelo Embedded em parceria com a Nokia e o INdT. Além do ESbox e PluThon, existe um pacote de conexão chamado *maemo-pc-connectivity*, o qual trás as ferramentas necessárias para estabelecer a conexaão entre o PC e o Internet Tablet. Atualmente, o PluThon está na versão 0.1.8 com cerca de 7 KLOC e possui versões para os sistemas operacionais Linux e Windows.

12.4.2 Arquitetura

Assim como o ESbox, a arquitetura do PluThon também é composta por duas camadas: *framework* e *plug-in*. A arquitetura do PluThon está ilustrada na Figura 12.4. Para realizar a transferência de scripts Python entre o computador e o Internet Tablet, o PluThon deve executar comandos que utilizam as ferramentas `ssh` e `scp`. Essas ferramentas estão presentes no sistema operacional (Linux, MacOS ou Windows). Assim, a camada *framework* encapsula estes comandos em instâncias de processos usando a linguagem Java. As saídas dos processos são direcionadas à camada *plug-in* para que o ambiente mantenha o programador informado sobre o estado de execução da aplicação. A camada *framework* também é responsável pela comunicação com a plataforma Eclipse e com os elementos da interface gráfica. Também é possível a adição de novos plug-ins ao PluThon.

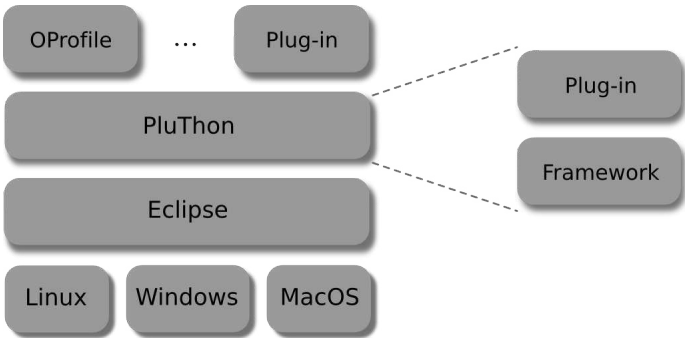

Figura 12.4: Arquitetura de ferramenta PluThon.

O PluThon é composto por 5 módulos, que assim como o ESbox, possuem funcionalida-

des específicas.

Core Implementa a camada `framework`. Outros elementos importantes para a ferramenta PluThon, tais como *natures* e *builders* estão presentes nesse módulo;

UI Módulo que contém as principais funcionalidades de interface gráfica, tais como *views*, *wizards* e *actions*. Grande parte da camada *plug-in* é implementada por este módulo;

Help Possui páginas html e imagens utilizadas na criação da documentação de auxílio ao desenvolvedor, integrada ao framework Eclipse;

Launch Módulo que implementa as funções que permitem a depuração remota de projetos PluThon;

Debug Implementa as funcionalidades para exeução de projetos PluThon.

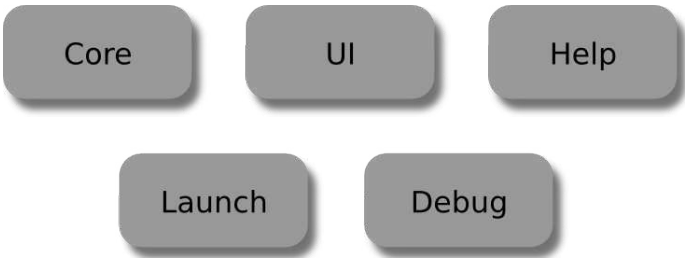

Figura 12.5: Módulos que compõe a ferramenta PluThon.

12.4.3 Requisitos

A ferramenta PluThon atualmente suporta os sistemas operacionais Linux, Windows e futuramente MacOS. A versão 0.1.8 da ferramenta requer o Eclipse 3.3.3.1 e PyDev 1.3.7 ou superior. Está licenciada sob EPL (Eclipse Public License) e os exemplos que podem ser utilizados pelos usuários estão licenciados sob LGPL (Lesse General Public License). Uma descrição mais completa da ferramenta, como também uma versao para instalação, pode ser encontrada em `http://pluthon.garage.maemo.org`.

12.4.4 Processos de Desenvolvimento

Como descrito no Capítulo 4, o processo de desenvolvimento de aplicações em Python para a plataforma maemo é um pouco diferente das aplicações em C/C++, pois não há a tarefa de compilação do código fonte. Dessa forma, para aplicações em Python, é necessário apenas codificar e testar diretamente, seja no ambiente desktop (caso o Scratchbox seja utilizado) ou Internet Tablet. O usuário da ferramenta PluThon não deseja instalar o ambiente Scratchbox, e, portanto, deve executar a aplicação diretamente no dispositivo. Contudo, com o auxílio da ferramenta, o programador deve apenas trabalhar no Eclipse sem se preocupar com a transferência do aplicativo para o Internet Tablet e sua execução posterior: todas essas tarefas são realizadas de maneira transparente.

A ferramenta PluThon está sendo utilizada cada vez mais por desenvolvedores Linux e Windows MacOS que esperam um ambiente ml para desenvolver suas aplicações Python para a plataforma maemo. Alguns comentários já foram postados em lista da comunidade bastante importantes [1], evidenciando a utilidade da ferramenta e também a boa aceitação entre os desenvolvedores veteranos de Python para maemo.

12.5 Conclusão

Ambientes integrados de desenvolvimento auxiliam bastante o programador durante do ciclo de criação de software. Um bom ambiente é aquele que possibilita a execução de todas as atividades envolvidas no desenvolvimento de software. Além de facilitar o aprendizado de novos desenvolvedores na plataforma maemo por apresentarem um ambiente gráfico de fácil utilização, as ferramentas ESbox e PluThon também favorecem a melhoria de qualidade da aplicação, pois diminuem o risco de implementar código com bugs. Além disso, as ferramentas trazem ganhos de tempo e esforço, pois o desenvolvedor não precisa se preocupar com a gerência correta dos diferentes ambientes envolvidos. O ESbox e o PluThon estão em plena fase de desenvolvimento, embora já acumulem muitas funcionalidades importantes. A tendência é de torná-los ferramentas cada vez mais completas para auxiliar o desenvolvedor nas tarefas mais variadas possíveis: desde criação de projeto, até uma completa análise da

[1] `http://maemogeek.blogspot.com/2008/04/pymaemo-talk-on-may-10th-in-florence-at.html`, `http://www.internettablettalk.com/2008/02/26/pluthon-scratchbox-less-development`

execução do aplicativo no Internet Tablet.

Capítulo 13

Outras Contribuições

Nos últimos anos, vários trabalhos têm buscado melhorias na execução de aplicações embarcadas através de mecanismos diferenciados de processamento de linguagens: compilação estática, interpretação, soluções híbridas e compilação Just-In-Time. Neste capítulo, é descrita uma outra contribuição deste trabalho: um compilador Just-In-Time para processadores baseados em ARM. A implementação de tal ferramenta é realizada através do framework de compilação LLVM, o qual permite a criação de compiladores, bem como uma série de otimizações e transformações na aplicação durante o seu ciclo de execução.

13.1 Introdução

Técnicas de tradução são utilizadas desde a década de 1950 para converter a representação de um programa em outra. A compilação estática e interpretação são duas das mais importantes técnicas de tradução para linguagens de programação. No primeiro caso, o código fonte é traduzido para o formato desejado aplicando-se uma série de análises e otimizações para se obter uma representação executável, geralmente em código *assembly*. No segundo caso, a interpretação, os mesmos tipos de análises usadas durante a compilação estática podem ser aplicadas. Entretanto, no lugar de gerar o código *assembly*, a execução é feita imediatamente [8].

Técnicas de compilação *Just-In-Time* (JIT), também conhecidas como tradução dinâmica, foram inicialmente desenvolvidas no início da década de 1960 [8]. Nós últimos anos, alguns trabalhos importantes foram desenvolvidos na área. A compilação JIT provê

benefícios de ambas as técnicas citadas: compilação estática e interpretação. Alguns compiladores JIT foram desenvolvidos para diversas linguagens de programação, tais como LC^2, Basic, FORTRAN, Smalltalk, Java, Perl 6, Python e C#. Entretanto, a maioria dos compiladores JIT foram concebidos para se tornarem o ambiente de execução de sua respectiva linguagem.

Um dos principais problemas na utilização de compiladores JIT está relacionado à falta de otimização no desempenho ou uso da memória. Em máquinas desktop, com poder de processamento e quantidade de memória abundantes, isto não é um problema. Entretanto, no contexto de sistemas embarcados, nos quais a quantidade de recursos disponíveis é limitada, o uso correto do processamento e memória são cruciais para uma execução satisfatória de qualquer aplicação. No processo de desenvolvimento de aplicações C/C++, é possível satisfazer tais requisitos. Contudo, apenas um número restrito de linguagens pode ser utilizado em sistemas embarcados. Além disso, caso o desenvolvedor tente portar aplicações desktop já existentes, deve verificar se o sistema embarcado utilizado possui suporte à linguagem em utilização.

O projeto *Low Level Virtual Machine* (LLVM) [48; 49; 50] é um framework de compilação desenvolvido para realizar otimizações de código durante o ciclo de vida de um programa. Ele provê um conjunto de instruções baseada em arquitetura RISC, além de ser independente de linguagem e de plataforma. O LLVM também disponibiliza compiladores JIT que traduzem e executam bytecodes LLVM em tempo de execução.

O framework LLVM é composto por um conjunto de componentes que facilitam o desenvolvimento de compiladores. É relativamente simples o processo de adição de um novo JIT no LLVM para uma determinada arquitetura: é necessária a implementação de um *seletor de instrução*, de um *emissor de código de máquina* e de um *descritor da arquitetura alvo*. Todas as análises e otimizações realizadas no código fonte já estão disponíveis no LLVM. O LLVM provê compiladores JIT para as arquiteturas 86, x86-64, PowerPC e Alpha. Neste trabalho, também foi desenvolvido o suporte JIT para a arquitetura baseada em *Advanced RISC Machine* (ARM) [79].

Esta contribuição foi focada no desenvolvimento de um compilador JIT para arquiteturas baseadas em ARM que executam o sistema operacional Linux. O principal objetivo é prover um JIT que execute de maneira eficiente sem consumir tantos recursos do sistema. Além

disso, o suporte a um grande número de linguagens de programação é o principal aspecto introduzido através desta contribuição, uma vez que o JIT executa bytecodes LLVM. Por essas razões, o framework de compilação LLVM é de grande valor para a implementação da solução proposta, permitindo uma abordagem independente de linguagem de programação e focada em detalhes específicos da arquitetura ARM.

Muitos dos processadores ARM são usados em pequenos dispositivos com recursos limitados. Compiladores JIT e interpretadores são raramente explorados em arquiteturas ARM, pois geralmente não produzem um bom resultado [20; 27]. Alguns trabalhos recentes implementaram máquinas virtuais Java para processadores ARM [52; 73]. Tais projetos procuram disponibilizar um executável de tamanho pequeno e com várias funcionalidades de uma máquina virtual Java para dispositivos móveis com Linux. A maioria das pesquisas focam na plataforma .NET e Java, e somente algumas poucas linguagens podem ser traduzidas e executadas por um compilador JIT e interpretadas em dispositivos ARM.

13.2 LLVM

O framework de compilação LLVM foi desenvolvido para otimização de código durante o ciclo de vida do programa. As otimizações são realizadas em tempo de compilação, tempo de ligação, tempo de execução e até mesmo no modo *offline* de maneira transparente para os programadores. Para obter esses diferentes níveis de otimização, um conjunto de instruções para máquinas RISC é disponibilizado para representar o código fonte. O conjunto de instruções é independente de linguagem e de arquitetura. Além disso, suporta tipos de dados e informação sobre o fluxo de controle. Além disso, a infra-estrutura do LLVM provê um *front-end* baseado em GCC para a linguagem C/C++ e um conjunto de *back-ends* estáticos para várias plataformas, incluindo x86, x86-64, PowerPC 32/64, ARM, IA-64, Alpha e SPARC. Há também um *back-end* que gera código C portável e compilador JIT (*Just-In-Time*) para x86, x86-64 e PowerPC 32/64.

O LLVM torna mais fácil a adição de suporte a uma nova linguagem ou arquitetura se comparado a outros compiladores existentes (por exemplo, GCC), através da separação explícita de *front-ends* (linguagens de programação) e *back-ends* (plataforma alvo) e da utilização de uma representação intermediária (RI) adequada. Mais especificamente, para adi-

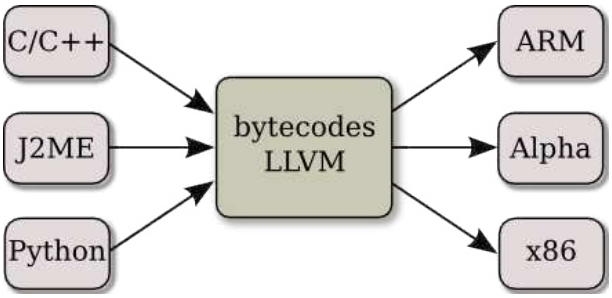

Figura 13.1: Cenário de utilização.

cionar suporte a uma nova linguagem é necessário implementar um novo *front-end* para a linguagem alvo e para adicionar suporte a uma nova arquitetura, basta implementar um novo *back-end*. Isto é possível graças ao fato de que a RI do LLVM é independente de hardware e da linguagem da aplicacção a ser compilada. Enquanto o *front-end* é responsável pela compilação do código fonte e tradução para a RI, o *back-end* transforma o código em RI para código de máquina, sendo posteriormente executado. Além disso, operações de otimização de código são também aplicadas. Um cenário típico é ilustrado na Figura 13.1.

A estratégia de compilação LLVM está ilustrada na Figura 13.2. Primeiramente, o código fonte é compilado e arquivos-objeto são gerados. Tais arquivos não possuem nenhum código de máquina, mas somente bytecode LLVM. Logo após, um ligador combina os arquivos-objeto e bibliotecas nativas, aplicando procedimentos de otimização, para produzir código executável, o qual pode ser transformado em código nativo ou bytecodes LLVM. É importante acrescentar que o JIT é usado neste caso. Finalmente, a aplicação pode ser executada para gerar traços de execução e informação de perfilamento que serão utilizados pelo otimizador de tempo de execução. A estratégia de compilação adotada não difere muito da tradicional, exceto nas fases que envolvem otimizações em tempo de execução e *offline*, as quais são opcionais.

A fase de otimização não depende da arquitetura alvo tampouco da linguagem utilizada na implementação. A linguagem tratada durante o processo de otimização é o conjunto de instruções LLVM. Isso significa que a adição de uma nova linguagem requer apenas a implementação de um novo *front-end*. Do mesmo modo, a adição de uma nova arquitetura requer somente a implementação de um novo *back-end*. A construção de *front-ends* e *back-ends* não são tarefas facilmente realizadas. Contudo, são bem mais simples de serem realizadas

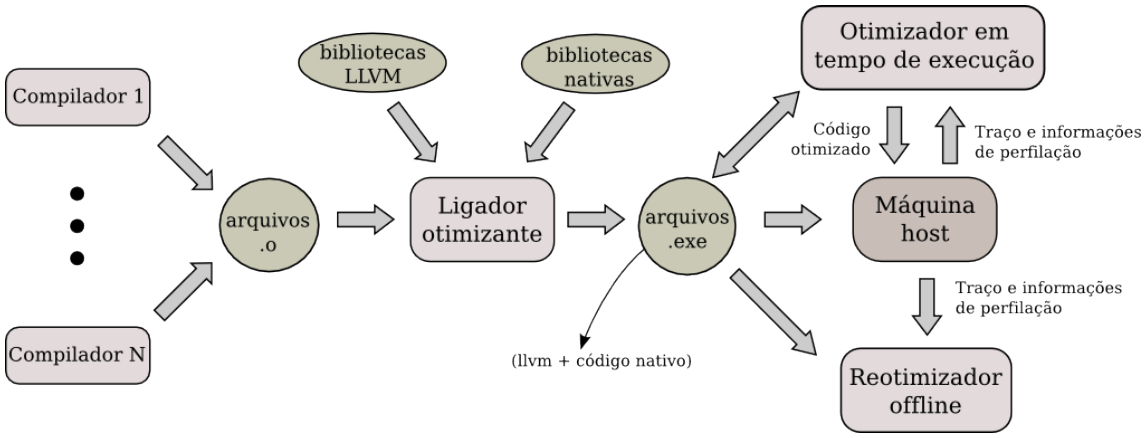

Figura 13.2: Estratégia de compilação do LLVM.

do que construir um compilador completo.

Para possibilitar o conjunto de análises e transformações de alto-nível, é necessário que a linguagem de bytecodes LLVM seja bem expressiva. Além diso, as otimizações realizadas pelo LLVM não devem ser muito diferentes das otimizações de código nativo [24]. Por outro lado, os bytecodes LLVM também devem ser de baixo nível o suficiente para permitir otimizações em tempo de execução sem oferecer muita carga para a aplicação.

13.2.1 Fases de geração de código

O framework LLVM possui um conjunto de componentes reusáveis que permitem a tradução de código em RI LLVM para código de máquina de uma determinada arquitetura alvo. O código resultante pode ser gerado em formato *assembly* ou em código binário de máquina. A primeira forma é obtida a partir de compiladores estáticos, enquanto que a segunda é resultante da utilização de compiladores JIT. As fases de geração de código consistem de:

1. Uma descrição abstrata da arquitetura alvo que captura aspectos peculiares de um dado processador;

2. Um conjunto de elementos abstratos que representam o código de máquina gerado para um processador alvo;

3. Um conjunto de algoritmos independentes de plataforma que são utilizados para geração de código nativo;

4. A implementação de interfaces de descrição específicas de arquitetura que são utilizadas para gerar o código de máquina correto para um dado processador.

O gerador de código consiste das fases ilustradas na Figura 13.3. No momento em que o gerador de código é executado, os bytecodes LLVM já foram previamente gerados. Na fase de *seleção de instruções*, os bytecodes LLVM são transformados em código de máquina de baixo nível. Embora o código inicial seja dependente da arquitetura, ele ainda é gerado em um formato de instruções de máquina abstrata e algumas fases independentes da arquitetura são usadas para essa tradução. Assim, um conjunto de otimizações em código no formato SSA [5] são aplicadas. Por exemplo, para eliminação de código inútil (que nunca será executado) e propagação de constantes. Assim como o código alvo está em formato SSA, as variáveis definidas estão armazenadas em registradores virtuais infinitos. Neste fase, é necessário restringir o número de registradores para usar somente a quantidade de registradores físicos disponível na arquitetura alvo. Este trabalho de escolha dos registradores físicos é executado na fase de alocação de registradores ilustrada na Figura 13.3.

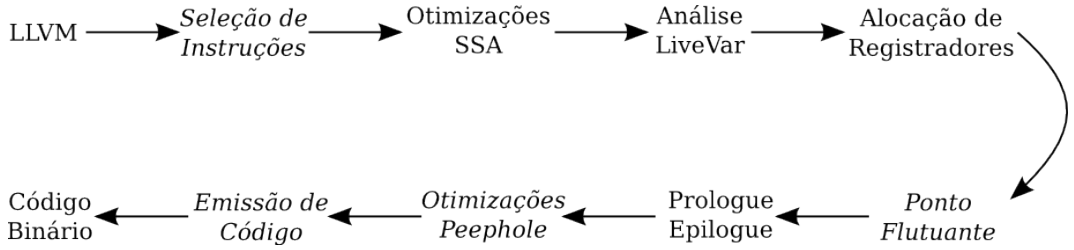

Figura 13.3: Fases de geração de código.

As fases de *prologue/epilogue* e *peephole* realizam algumas transformações para otimizar o código. Finalmente, a fase de *emissão de código* gera código de máquina ou *assembly*.

13.3 Compilador JIT para ARM

Bons compiladores JIT devem conhecer muitos detalhes da plataforma para os quais foram desenvolvidos, incluindo memória, registradores e utilização da pilha. O tratamento correto desses aspectos pelo JIT impacta diretamente em seu desempenho. Nesta seção, discute-se a implementação do suporte JIT para o *back-end* ARM do framework LLVM, realizada durante este trabalho.

13.3.1 Compiladores JIT e processadores baseados em ARM

Um compilador JIT deve produzir código rápido [20; 27], e é utilizado para melhorar a eficiência de aplicações. A compilação estática e a interpretação são duas técnicas de tradução comumente utilizadas. Ao utilizar um JIT, o usuário pode se beneficiar das vantagens de ambas as abordagens [8].

Programas compilados estaticamente executam de maneira mais rápida do que aqueles que são interpretados. Geralmente, um conjunto de otimizações são aplicadas durante a compilação para obter um código menor e mais rápido. Um compilador JIT não deve afetar de forma negativa a execução de um programa. Linguagens utilizadas para representar programas interpretados geralmente estão em um nível de abstração maior do que o código de máquina, e também agregam mais semântica. Dessa forma, o código obtido é menor. Assim, as linguagens utilizadas para representar os programas interpretados tendem a ser portáveis e somente o interpretador é dependente da plataforma. Finalmente, os interpretadores possuem acesso a informações disponíveis em tempo de execução, por exemplo, o fluxo de controle e as especificações da arquitetura alvo. Como conseqüência, o uso de JIT permite a geração de um código menor e mais rápido.

A utilização de uma linguagem de alto nível para representar o código permite a sua execução em todas as plataformas para as quais o JIT foi portado. Nenhum suporte para processadores ARM foi implementado até o momento. Os primeiros compiladores JIT apresentaram resultados de desempenho insatisfatórios, mesmo para máquinas desktops. Entretanto, a necessidade por JITs mais rápidos, especialmente para a linguagem Java, tem encorajado pesquisas na área e mais esforço tem sido aplicado [8].

Os processadores ARM foram desenvolvidos com o intuito de obter um hardware de tamanho reduzido com alto desempenho [79]. Por conta do tamanho reduzido, baixo consumo de energia é outra característica da plataforma. Processadores ARM são comumente utilizados em telefones celulares, handhelds, Internet Tablets e sistemas elétricos automotivos. Desde o lançamento do ARM6^{TM}, mais de 10 bilhões de processadores foram produzidos.

O processador possui funcionalidades de uma arquitetura RISC típica, por exemplo, instruções de tamanho fixo e do tipo *load/store*. Também provê funções importantes que permitem um conjunto de otimizações, código de tamanho reduzido e baixo consumo de

memória. Ele permite modos de endereçamento de auto-incrementa e auto-decrementa (ao executar a função o valor dos registradores é incrementado ou decrementado), os quais facilitam a otimização de laços do programa. Além disso, as instruções *load/store* de múltiplos [1] são usadas para salvar/recuperar o contexto e maximizar a taxa de transferência. As instruções ARM podem ser condicionalmente executadas, o que permite execução eficiente de código. Finalmente, algumas instruções (por exemplo, ADD e SUB) modificam registradores da Unidade de Aritmétrica e Lógica (UAL) e os manipula para otimizar o seu uso.

13.3.2 Utilizando o JIT LLVM

Quando o LLVM é compilado para uma arquitetura específica, o JIT também é compilado se existir implementação para o alvo. O JIT LLVM é invocado através da ferramenta `lli`[2] ou através da implementação de programas que geram a função e as carrega já na memória.

O *lli* executa programas compilados para o formato de bytecodes LLVM. A aplicação é executada usando o JIT, caso esteja disponível. Caso contrário, um interpretador é usado. Também é possível implementar um programa que cria uma instância do JIT e executa o bytecode da aplicação. O programa usa o JIT para gerar todas as instruções nativas da aplicação. O resultado da última abordagem é um código geralmente mais rápido pelo fato de não ser necessário carregar os bytecodes LLVM. Entretanto, essa solução demanda trabalho extra, sendo bastante custosa para softwares maiores.

O código a seguir exemplifica como a API LLVM pode ser utilizada para gerar instruções. No método `CreateFibFunction`, é criada uma função que implementa a versão recursiva do algoritmo de Fibonacci. O JIT é declarado e inicializado, e a função Fibonacci é instanciada na função `main`. Finalmente, a função que implementa o algoritmo de Fibonnaci (`func()`)é invocada com um dado parâmetro.

```
1 static Function *CreateFibFunction(Module *M) {
2   // Cria a funcao fib e a insere no modulo M.
3   // Esta funcao retorna um int e recebe um int como parametro.
4   Function *FibF = cast<Function>
5       (M->getOrInsertFunction("fib", Type::Int32Ty, \
6                                   Type::Int32Ty, (Type *)0));
```

[1] Permitem salvar/recuperar o conteúdo de vários registradores ao mesmo tempo

[2] `lli` LLVM tool. `http://www.llvm.org/cmds/lli.html`.

```
7     ...
8
9     // cria fib(x-1)
10    Value *Sub = BinaryOperator::createSub(ArgX, One, "arg", \
11                                                        RecurseBB);
12    Value *CallFibX1 = new CallInst(FibF, Sub, "fibx1", RecurseBB);
13
14    ...
15
16    return FibF;
17  }
18
19  int main(int argc, char **argv) {
20    ...
21
22    // Instancia um modulo para inserir uma funcao
23    Module *M = new Module("test");
24
25    // Instancia a funcao "fib"
26    Function *FibF = CreateFibFunction(M);
27
28    // Instancia o JIT
29    ExistingModuleProvider *MP = new ExistingModuleProvider(M);
30    ExecutionEngine *EE = ExecutionEngine::create(MP, false);
31
32    ...
33
34    // Invoca a funcao de Fibonacci com o argumento n:
35    std::vector<GenericValue> Args(1);
36    Args[0].IntVal = n;
37    int (*func)(int) = (int(*)(int))EE->getPointerToFunction(FibF);
38
39    ...
40
41    int r = func(n);
42
43    ...
```

```
44
45    return 0;
46  }
```

13.3.3 Questões relativas à implementação

O framework LLVM disponibiliza uma API flexível para implementar o JIT para qualquer arquitetura. Para possibilitar a adição de novas arquiteturas alvo, o JIT LLVM gera código através de diferentes estágios: otimização, transformação e geração de código. Na Figura 13.3, o fluxo da tarefa de geração de código é exibido. Tarefas específicas de plataforma estão em itálico, enquanto que as demais são independentes de plataforma.

Uma vez que o framework de JIT LLVM não é bem documentado, a adição de novas targets é uma tarefa complexa para o desenvolvedor que não está familiarizado com alguns detalhes técnicos. Entretanto, a comunidade LLVM é bastante envolvida no projeto e a falta de documentação é compensada pela colaboração dos contribuidores LLVM. Uma vez que esse problema é resolvido, a adição de um novo suporte JIT se torna mais fácil. As arquiteturas suportadas são x86, x86-64, PowerPC, Alpha, e agora ARM. O problema mais dífícil relativo à compilação JIT de bytecodes LLVM está relacionado à representação eficiente e atributos específicos do alvo, arquivo de registradores e conjunto de instruções [51].

Basicamente, o emissor de código de máquina é um módulo que modifica a RI, chamado de pass, os quais realizam as transformações e otimizações, e também geram resultados de análises que são utilizadas nas transformações. Os *passes* LLVM são subclasses da classe `Pass` [2]. O framework LLVM disponibiliza diferentes *passes*, cada um com uso em contextos específicos. Por exemplo, `ImmutablePass` é utilizado por *passes* que não precisam ser executados, não modificam o estado do programa e nunca precisam ser atualizados. Os *passes* LLVM podem acessar a RI LLVM em diferentes níveis de granularidade. O compilador JIT, implementado no arquivo `ARMCodeEmitter.cpp`, é subclasse de um `MachineFunctionPass`, o qual iterage sobre os blocos de função de máquina e também sobre instruções de máquina.

O *pass* iterage sobre cada instrução de máquina e gera o código binário respectivo. As instruções são montadas e carregadas na memória principal para serem posteriormente exe-

cutadas [51]. O conjunto de instruções de máquina do processador precisa ser descrito e transformado em seqüência de dados binários que serão utilizados para a montagem das instruções. As instruções suportadas pelo emissor de código, JIT ou *assembler* são descritas no arquivo `ARMInstrInfo.td`, o qual consiste basicamente de uma tabela que define o conjunto de instruções.

O processo de geração de instruções de máquina em formato binário seria mais complexo e difícil caso o mecanismo de descrição de instruções não fosse adotado. Dessa forma, o emissor de código de máquina é baseado na descrição do conjunto de instruções da arquitetura alvo. O Conjunto de Instruções da Arquitetura (*Instruction Set Architecture* - ISA) alvo é descrita através de uma linguagem especial que permite separar instruções em classes baseadas em características comuns, como número de operadores, por exemplo. Além disso, tal abordagem permite que o conjunto de instruções da arquitetura alvo seja facilmente estendida e compreendida. Entretanto, algumas limitações da arquitetura ARM que impactaram na implementação do JIT tiveram que ser resolvidas. Tais limitações são discutidas nas próximas seções.

Diferentes formatos de instrução

As instruções ARM possuem um tamanho fixo de 32-bits. Tal característica facilita a transformação das instruções de máquina em formato binário, uma vez que o tamanho é previamente conhecido. Entretanto, o número, a localização e o tamanho de campos da instruções podem variar de uma instrução para outra. O único campo em comum é o campo *condicional*, que é usado para determinar a execução condicional da instrução. Instruções ARM não possuem um *opcode* (*operation code*) explícito que determina qual a operação a ser realizada. Dessa forma, é difícil de identificar qual é a instrução de máquina corrente. Além disso, o número de operadores pode mudar de uma instrução para a outra. Essas mudanças são exemplificadas na Figura 13.4, na qual são ilustradas duas classes de instrução ARM: a) processamento de dados com *shift* imediato e b) *load/store* com *offset* imediato. Verifica-se que ambas as classes de instruções não possuem o campo *opcode*, mas somente o campo *condicional* e o espaço para os registradores (Rn e Rd) são comuns. É possível argumentar que no primeiro caso, há um campo de opcode. Entretanto, esse campo é referente ao opcode que identifica uma instrução load/store, e não é um opcode único que identifica a instrução

dentre várias no conjunto de instruções ARM.

Dessa forma, é necessário utilizar uma abordagem mais abstrata para classificar as instruções. Cinco diferentes classes são definidas: i) processamento de dados; ii) *load/store*; iii) desvio; iv) multiplicação, e; v) instruções de aritmética diversas [79; 82]. Há mais formatos disponíveis, tais como semáforo e instruções de co-processamento. Porém, elas não são implementadas no JIT para ARM. Os formatos das instruções são implementados como flags que devem ser modificados de acordo com o arquivo `ARMInstrInfo.td`. As flags são enumeradas no arquivo `ARMInstrInfo.h`.

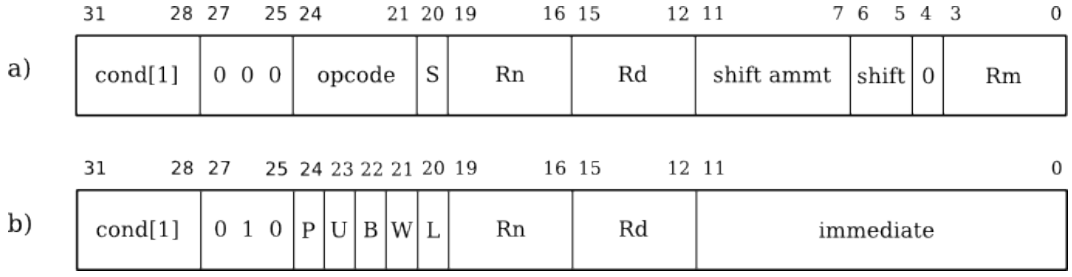

Figura 13.4: Exemplo de formatos de instrução do ARM.

Realocações

O JIT do LLVM possui uma estratégia de compilação tardia (*lazy*). Cada instrução de máquina é gerada somente se a função for invocada. O LLVM adiciona âncoras em cada chamada a função, as quais são usadas para encontrar o endereço correto do bloco de instruções de máquina da função na memória principal. Este cálculo é feito baseando-se no endereço de retorno da chamada da função, a qual foi empilhada previamente.

O endereço de retorno é buscado em uma tabela com o intuito de determinar a função que deve ser chamada daquela localização. Se o código da instrução de máquina não tiver sido gerado ainda para aquela função, o gerador de código é invocado. Então, o contador do programa (PC) é atualizado para apontar para o endereço real da função. Este cálculo realizado para determinar o endereço é chamado de realocação. As realocações ARM são definidas no arquivo `ARMRelocations.h`. Dois tipos estão definidos: `reloc_arm_relative` e `reloc_arm_branch`.

A realocação do tipo `reloc_arm_relative` é usada para identificar endereços não-globais, ou seja, símbolos externos, endereços de *pool* de constantes e endereços de tabela.

Neste caso, é necessário calcular o valor correto do PC subtraindo o endereço de retorno do endereço alvo para formar um offset. A realocação `reloc_arm_branch` encontra o valor correto para ser substituído no campo *signed_immed_24* das instruções de desvio ARM [79]. As operações realizadas para encontrar os endereços de realocação corretos estão no arquivo `ARMJITInfo.cpp`.

13.4 Conclusão

Neste capítulo, foi explorada a implementação do suporte de um compilador JIT para processadores ARM usando o framework de compilação LLVM. As principais características da arquitetura ARM, tais como formatos de instruções de máquina, foram consideradas para obter um resultado com bom desempenho. O framework LLVM provê pontos que permitem estender suas funcionalidades através da adição de suporte para novas linguagens e arquiteturas. Também é possível implementar e integrar novas otimizações e transformações de código no LLVM. O gerador de código LLVM torna possível a tradução de código fonte em instruções de máquinas em formato binário e então o JIT as carrega para a memória. O JIT apresentado foi implementado durante este trabalho, sendo uma importante contribuição para o projeto LLVM, bem como para pesquisas em métodos de compilação para sistemas embarcados com Linux.

Capítulo 14

Conclusões e Trabalhos Futuros

A crescente popularização de dispositivos móveis tem aumentado o interesse da indústria em produzir novos produtos que atendam aos requisitos dos usuários, cada vez mais exigentes. Aliado a esse crescimento no consumo, e à adoção de grandes empresas por produtos baseados em soluções *open source*, a Nokia lançou os produtos da linha Internet Tablet, que utilizam a plataforma maemo, baseada no sistema operacional Linux. Documentação sobre a plataforma maemo pode ser encontrada no site [58] e outras fontes externas. Contudo, estas informações estão organizadas de maneira dispersa, sem uma ordem que facilite o entendimento por parte do programador. Além disso, não existem ambientes gráficos que auxiliem na realização das diversas tarefas envolvidas no processo de desenvolvimento de aplicações maemo, por exemplo, codificação, documentação e compilação. Assim, a comunidade maemo carece de i) um material didático com informações dispostas de maneira organizada, sistematizada e concisa para ajudar no aprendizado da plataforma maemo (arquitetura e desenvolvimento) e ii) ambientes de desenvolvimento com funcionalidades que facilitem a implementação, compilação, execução e testes (depuração e testes de unidade) de aplicações maemo.

Neste trabalho foi apresentado a definição, organização e apresentação de um livro sobre a plataforma maemo direcionado para profissionais que desejem aprender a desenvolver aplicações para a plataforma, bem como para aqueles que buscam um guia de referência sobre aspectos mais avançados. O livro reúne informações importantes dispostas em uma maneira sistematizada, facilitando a pesquisa por conteúdo e também a aprendizagem sobre a plataforma, uma vez que as informações estão dispostas de maneira a melhorar o entendi-

mento. O material contempla vários aspectos de desenvolvimento, por exemplo, descrição e histórico da plataforma maemo, interface gráfica, multimídia, comunicação entre aplicações e conectividade. Todas as informações usadas para escrita desse material são oriundas de livros, artigos (técnicos e científicos), documentação existente em [58] e em outros sites. Além disso, experiências de alguns desenvolvedores de aplicações maemo foram também consideradas.

Além do material sobre arquitetura e desenvolvimento da plataforma maemo, também foram descritas duas ferramentas bastante importantes para o desenvolvimento de aplicações maemo: ESbox [22; 40] e PluThon [23]. Tais ambientes de desenvolvimento integrado foram desenvolvidos no Laboratório Emdedded, da Universidade Federal de Campina Grande, e foram adotados pela plataforma maemo como ferramentas oficiais de desenvolvimento para a plataforma, integrando o projeto *IDE Integration* [56]. As ferramentas geradas oferecem funcionalidades que facilitam o processo de desenvolvimento de aplicações maemo, tais como editor sensível à sintaxe, compilador GCC integrado, depuração GDB integrada (visualização de registradores, mapa de memória, sinais, módulos carregados, etc.) e até mesmo ferramentas de perfilação (Valgrind e OProfile). Além disso, permitem a execução e depuração no Internet Tablet.

Também foi descrita neste trabalho uma outra contribuição importante realizada durante este trabalho: o suporte de um compilador JIT (Just-In-Time) para permitir a execução de aplicações em formato bytecodes LLVM em qualquer processador baseado em ARM que tenha Linux como sistema operacional.

Os resultados foram bastante motivantes. Em relação à proposta do livro, representantes das instituições responsáveis pela publicação de livros sobre plataformas Nokia se mostraram bastante motivados em relação à idéia. No tocante às ferramentas de desenvolvimento, o impacto sobre a comunidade foi positivo. O número de casos de utilização das ferramentas cresce acentuadamente, bem como a aceitação por parte dos desenvolvedores.

Como trabalhos futuros, o material produzido deverá ser incrementado com mais exemplos e exercícios, tornando-o também uma referência de livro para aprendizado da plataforma maemo. O conteúdo e organização do livro a ser publicado está ilustrado na Figura 14.1. O livro é composto por duas partes: a primeira apresenta informações introdutórias sobre o maemo, além de exemplos de aplicações que exploram os recursos da plataforma (camera,

som, conectividade); a segunda parte é um guia para busca por detalhes sobre os diversos subsistemas da plataforma, tais como conectividade e multimídia. Esforços serão aplicados para que o livro também seja traduzido para o idioma inglês para posterior publicação. Dessa forma, toda a comunidade maemo será beneficiada. Em relação às ferramentas desenvolvidas (ESbox e PluThon), além da manutenção do código já existente, há atividades de refatoração que devem ser realizadas para obter um código de maior qualidade, possibilitando que outros desenvolvedores da comunidade também contribuam mais facilmente com as ferramentas. Há, ainda, a adição de novas funcionalidades: análise de código baseada em fluxo de dados e controle; atualização dinâmica do ambiente de desenvolvimento instalado; geração de pacotes de maneira transparente para o desenvolvedor para posterior implantação no ambiente; dentre outros. A implementação destas funcionalidades em ambas as ferramentas pode tornar os ambientes cada vez mais completos e seguros para o desenvolvimento de aplicações maemo.

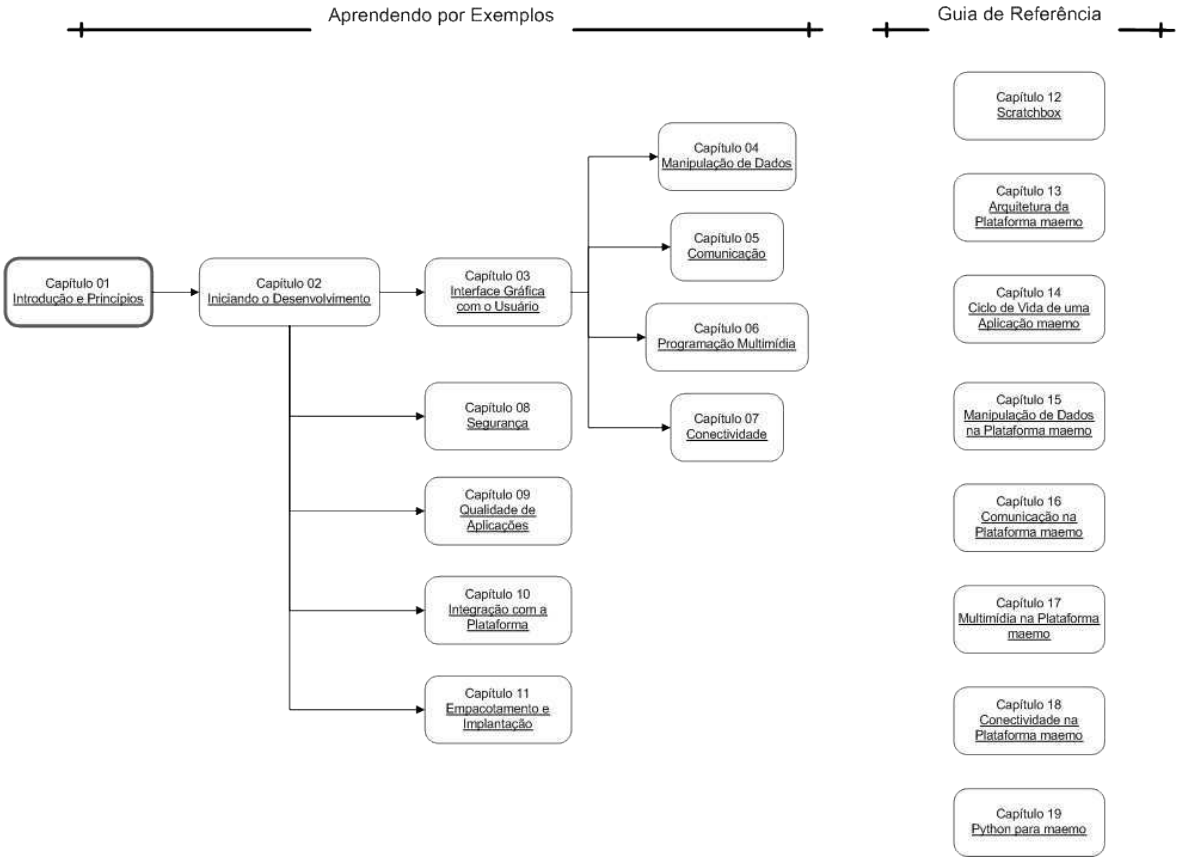

Figura 14.1: Conteúdo e organização do livro proposto.

Bibliografia

[1] curl and libcurl. Disponível on-line em `http://curl.haxx.se/`. Último acesso em 28/06/2008.

[2] The LLVM compiler infrastructure project. Disponível on-line em `http://www.llvm.org`.

[3] Oprofile - a system profiler for linux. Disponível on-line em `http://oprofile.sourceforge.net/`. Último acesso em 29/06/2008.

[4] GCC GNU Compiler. Disponível on-line em `http://gcc.gnu.org`, 2008. Last access on 03/24/2008.

[5] Andrew W. Appel. *Modern Compiler Implementation in C: Basic Techniques*. Cambridge University Press, New York, New York, EUA, 1997.

[6] ARM. *ARM*. Disponível on-line em `http://www.arm.com`, 2008. Last access on 03/14/2008.

[7] John Arthorne and Chris Laffra. *Official Eclipse 3.0 FAQS*. Addison Wesley, Boston, Massachusetts, EUA, 2004.

[8] John Aycock. A brief history of just-in-time. *ACM Comput. Surv.*, 35(2):97–113, 2003.

[9] Daniel J. Barrett and Richard E. Silverman. *SSH, The Secure Shell: The Definitive Guide*. O'Reilly & Associates, Inc., Sebastopol, Califórnia, EUA, 2001.

[10] Daniel Bartholomew. Qemu: a multihost, multitarget emulator. *Linux J.*, 2006(145):3, 2006.

[11] R. Bell and D. Sharon. Tools to engineer new technologies into applications. *Software, IEEE*, 12(2):11–16, Mar 1995.

[12] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 41–41, Berkeley, Califórnia, EUA, 2005. USENIX Association.

[13] Christian Benvenuti. *Understanding Linux Network Internals*. O'Reilly Media, Inc., 2005.

[14] Jasmin Blanchette and Mark Summerfield. *C++ GUI Programming with Qt 4*. Prentice Hall PTR, Upper Saddle River, New Jersey, EUA, 2006.

[15] BlueZ. Bluez - official linux bluetooth protocol stack. Disponível on-line em `http://www.bluez.org/`. Último acesso em 28/06/2008.

[16] Douglas Boling. *Programming Microsoft Windows CE*. Microsoft Press, Redmond, WA, EUA, 1998.

[17] Jennifer Bray and Charles Sturman. *Bluetooth: Connect Without Cables*. Prentice Hall PTR, Upper Saddle River, New Jersey, EUA, 2000.

[18] Eric Clayberg and Dan Rubel. *Eclipse: Building Commercial-Quality Plug-ins*. Addison Wesley, Boston, Massachusetts, EUA, 2004.

[19] Jon Crowcroft and Iain Phillips. *TCP/IP and Linux protocol implementation: systems code for the Linux Internet*. John Wiley & Sons, Inc., New York, New York, EUA, 2002.

[20] Jason Domer, Murthi Nanja, Suresh Srinivas, and Bhaktha Keshavachar. Comparative performance analysis of mobile runtimes on Intel XScale®technology. In *IVME '04: Proceedings of the 2004 workshop on Interpreters, virtual machines and emulators*, pages 51–57, New York, New York, EUA, 2004. ACM.

[21] Leigh Edwards and Richard Barker. *Developing Series 60 Applications: A Guide for Symbian OS C++ Developers*. Pearson Higher Education, 2004.

[22] Laboratório Embedded. Esbox project. Disponível on-line em `http://esbox.garage.maemo.org`. Último acesso em 27/06/2008.

[23] Laboratório Embedded. Pluthon project. Disponível on-line em `http://pluthon.garage.maemo.org`. Último acesso em 27/06/2008.

[24] Michael Engel and Bernd Freisleben. Using a low-level virtual machine to improve dynamic aspect support in operating system kernels. In *Proceedings of the 4th AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS)*, Chicago, USA, March 2005.

[25] Gregor Engels, Wilhelm Schäfer, Robert Balzer, and Volker Gruhn. Process-centered software engineering environments: academic and industrial perspectives. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 671–673, Washington, Distrito de Columbia, EUA, 2001. IEEE Computer Society.

[26] Enlightenment. Enlightenment - the beauty in your fingertips. Disponível on-line em `http://web.enlightenment.org/p.php?p=index\&l=`. Último acesso em 30/06/2008.

[27] M.A. Ertl and D. Gregg. Retargeting jit compilers by using c-compiler generated executable code. *Parallel Architecture and Compilation Techniques, 2004. PACT 2004. Proceedings. 13th International Conference on*, pages 41–50, 29 Sept.-3 Oct. 2004.

[28] Nokia Europe. *OS2008 - Devices.* Disponível on-line em `http://europe.nokia.com/A4579470`. Último acesso em 27/06/2008.

[29] Freedesktop. pkg-config. Disponível on-line em `http://pkg-config.freedesktop.org/`, note = Último acesso em 28/06/2008,.

[30] Freedesktop. Xephyr. Disponível on-line em `http://freedesktop.org/wiki/Software/Xephyr`. Último acesso em 27/06/2008.

[31] freedesktop.org. D-bus. Disponível on-line em `http://dbus.freedesktop.org/`. Último acesso em 01/07/2008.

[32] GNOME. Gconf configuration system. Disponível on-line em `http://www.gnome.org/projects/gconf/`, note = Último acesso em 30/06/2008,.

[33] GNOME. Gnomevfs - filesystem abstraction library. Disponível on-line em `http://library.gnome.org/devel/gnome-vfs-2.0/2.22/`. Último acesso em 28/06/2008.

[34] Mark G. Graff and Kenneth R. Van Wyk. *Secure Coding: Principles and Practices*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2003.

[35] John Shapley Gray. *Interprocess communications in UNIX: the nooks and crannies*. Prentice-Hall, Inc., Upper Saddle River, New Jersey, EUA, 1996.

[36] Arthur Griffith. *GCC: The Complete Reference*. McGraw-Hill Professional, 2002.

[37] GStreamer. Gstreamer: open source multimedia framework. Disponível on-line em `http://www.gstreamer.net/`. Último acesso em 27/06/2008.

[38] Richard Harrison and Phil Northam, editors. *Symbian OS C++ for Mobile Phones*. John Wiley & Sons, Inc., New York, New York, EUA, 2003.

[39] Raul Herbster, Hyggo Oliveira, Angelo Perkusich, and Dalton Guerrero. Integrating open source tools for developing embedded linux applications. In *Proceedings of the Workshop on Free Software, 7th Internacional Forum on Free Software*, Porto Alegre, Rio Grande do Sul, Brasil, 2006. Sociedade Brasileira de Computação. International Track.

[40] Raul Herbster, Hyggo Oliveira, Angelo Perkusich, and Dalton Guerrero. Esbox: uma ferramenta para o desenvolvimento de aplicações para linux embarcado. In *Proceedings of the Workshop on Free Software, 9th Internacional Forum on Free Software*, Porto Alegre, Rio Grande do Sul, Brasil, 2008. Sociedade Brasileira de Computação.

[41] Craig Hollabaugh. *Embedded Linux: Hardware, Software, and Interfacing*. Addison Wesley, 2003.

[42] Michael Howard and Steve Lipner. *The Security Development Lifecycle*. Microsoft Press, Redmond, WA, EUA, 2006.

[43] Loki Software Inc. *Programming Linux Games*. Colaborador John R. Hall.

[44] Texas Instruments. High-performance: Omap2420, May 2007.

[45] M. Tim Jones. *GNU/Linux Application Programming*. Charles River Media, Inc., Rockland, Massachusetts, EUA, 2004.

[46] Mik Kersten and Gail C. Murphy. Using task context to improve programmer productivity. In *SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 1–11, New York, New York, EUA, 2006. ACM.

[47] Andrew Krause. *Foundations of GTK+ Development*. Apress, Berkely, Califórnia, EUA, 2007.

[48] Chris Lattner. LLVM: An infrastructure for multi-stage optimization. Master's thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, Urbana, Illinois, EUA, December 2002.

[49] Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, Califórnia, EUA, March 2004.

[50] Chris Lattner and Vikram Adve. The llvm compiler framework and infrastructure tutorial. In *LCPC'04 Mini Workshop on Compiler Research Infrastructures*, West Lafayette, Indiana, EUA, September 2004.

[51] Chris Lattner, Misha Brukman, and Brian Gaeke. *Jello: a retargetable Just-In-Time compiler for LLVM bytecode*. Disponível on-line em `http://www.arm.com`, 2003. Último acesso em 29/06/2008.

[52] Robert Lougher. Jamvm. Disponível on-line em `http://jamvm.sourceforge.net/`. Último acesso em 02/07/2008.

[53] Saulo Oliveira Dornellas Luiz, Jayarama Sundar Santana, Genildo de Moura Vasconcelos, Angelo Perkusich, Antonio Marcus Nogueira Lima, and Marcos Ricardo Morais.

A methodology to deploy applications on the dual-core omap platform. In *XVI Congresso Brasileiro de Automática, 2006, Salvador: Anais do XVI Congresso Brasileiro de Automática*, pages 3360–3365, 2006.

[54] Mark Lutz. *Programming Python: Object-Oriented Scripting*. O'Reilly & Associates, Inc., Sebastopol, Califórnia, EUA, 2001. Foreword By-Guido Van Rossum.

[55] Mark Lutz. *Learning Python*. O'Reilly & Associates, Inc., Sebastopol, Califórnia, EUA, 2003.

[56] Maemo. Ide integration. Disponível on-line em `http://maemo.org/development/documentation/ide\_integration/`. Último acesso em 27/06/2008.

[57] Maemo. Libosso. Disponível on-line em `http://maemo.org/api_refs/4.0/libosso/index.html`. Último acesso em 28/06/2008.

[58] maemo.org. maemo.org: Maemo™is the development platform for internet tablets. Disponível on-line em `http://www.maemo.org`. Último acesso em 01/07/2008.

[59] Veli Mankinen and Valtteri Rahkonen. *Cross-Compiling Tutorial with Scratchbox*. Disponível on-line em `http://www.scratchbox.org/documentation/docbook/tutorial.html`, 2004.

[60] Norman S. Matloff and Carsten Wartmann. *The Art of Debugging with GDB/DDD: For Professionals and Students*. No Starch Press, São Francisco, Califórnia, EUA, 2003.

[61] Jeff McAffer and Jean-Michel Lemieux. *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java(TM) Applications*. Addison-Wesley Professional, 2005.

[62] Robert Mecklenburg. *Managing Projects with GNU Make (Nutshell Handbooks)*. O'Reilly Media, Inc., 2004.

[63] S. Meyers. Difficulties in integrating multiview development systems. *Software, IEEE*, 8(1):49–57, Jan 1991.

[64] Movial. Scratchbox - sbrsh. Disponível on-line em `http://www.scratchbox.org/documentation/user/scratchbox-1.0/html/sbrsh.html`. Último acesso em 28/06/2008.

[65] Movial. Scratchbox Toolchain. Disponível on-line em `http://www.scratchbox.org`, 2008. Último acesso em 03/24/2008.

[66] mozdev.org. Petname project. Disponível on-line em `http://petname.mozdev.org/`. Último acesso em 02/07/2008.

[67] John Murray. *Inside Microsoft Windows CE*. Microsoft Press, Redmond, WA, EUA, 1998.

[68] Nicholas Nethercote and Julian Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. In *PLDI '07: Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation*, pages 89–100, New York, New York, EUA, 2007. ACM.

[69] Nokia. Forum nokia. Disponível on-line em `http://forum.nokia.com`. Último acesso em 27/06/2008.

[70] Openbossa. Canola. Disponível on-line em `http://openbossa.indt.org.br/canola/`. Último acesso em 28/06/2008.

[71] Openbossa. Python para maemo. Disponível on-line em `http://pymaemo.garage.maemo.org/`. Último acesso em 28/06/2008.

[72] Havoc Pennington. *GTK+/Gnome Application Development*. New Riders Publishing, Thousand Oaks, Califórnia, EUA, 1999. Prefácio por Miguel de Icaza.

[73] Evolvis Project. Jalimo. Disponível on-line em `http://www.jalimo.org`. Último acesso em 02/07/2008.

[74] GNU Project. Gnu c library. Disponível on-line em `http://www.gnu.org/software/libc/`. Último acesso em 30/06/2008.

[75] Arnold Robbins. *GDB Pocket Reference (Pocket Reference (O'Reilly))*. O'Reilly Media, Inc., 2005.

[76] John Rose and Stephanie Provines. *The Book of WI-FI: Install, Configure, and Use 802.11b Wireless Networking*. No Starch Press, São Francisco, Califórnia, EUA, 2003.

[77] Robert W. Scheifler, James Gettys, Al Mento, and Donna Converse. *X Window System: Extension Library : X Version 11, Release 6 and 6.1*. University of Michigan, 1997.

[78] Robert Seacord. Secure coding in c and c++: Of strings and integers. *IEEE Security and Privacy*, 4(1):74, 2006.

[79] David Seal. *ARM Architecture Reference Manual*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[80] Sherry Shavor, Jim D'Anjou, Scott Fairbrother, Dan Kehn, John Kellerman, and Pat McCarthy. *The Java Developer's Guide to Eclipse*. Addison-Wesley Longman Publishing Co., Inc., Boston, Massachusetts, EUA, 2003.

[81] Abraham Silberschatz and Peter Baer Galvin. *Operating System Concepts*. John Wiley & Sons, Inc., New York, NY, USA, 2000.

[82] Andrew Sloss, Dominic Symes, and Chris Wright. *ARM System Developer's Guide: Designing and Optimizing System Software*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[83] Jo Stichbury, editor. *Symbian OS Explained: Effective C++ Programming for Smartphones*. John Wiley & Sons, Inc., New York, New York, EUA, 2004.

[84] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall PTR, Upper Saddle River, New Jersey, EUA, 2001.

[85] ThoughtFix. 3rd anniversary special: A timeline from 2005 to 2008. Disponível on-line em `http://tabletblog.com/2008/05/3rd-anniversary-special-timeline-from.html`. Último acesso em 30/06/2008.

[86] Video4Linux. Video4linux. Disponível on-line em `http://www.video4linux.net/`. Último acesso em 28/06/2008.

[87] Jon Viega, Pravir Chandra, and Matt Messier. *Network Security with Openssl*. O'Reilly & Associates, Inc., Sebastopol, Califórnia, EUA, 2002.

[88] Matthias Warkus. *The Official GNOME 2 Developer's Guide*. No Starch Press, São Francisco, Califórnia, EUA, 2004.

[89] David A. Wheeler. *Secure Programming for Linux and Unix HOWTO*. 2003. Disponível on-line `http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/index.html`.

[90] W. Wolf. *Computer as Component: Principles of Embedded Systems Design.* Morgan kaufmann, 5 edition, 2005.

[91] Wayne Wolf. *Computer as Components: principles of embedded computing system design*. Morgan Kaufmann, São Francisco, Califórnia, EUA, 2005.

[92] Karim Yahgmour. *Building Embedded Linux Systems*. O'Reilly, California, EUA, 2003.

[93] Zhihui Yang and M. Jiang. Using eclipse as a tool-integration platform for software development. *Software, IEEE*, 24(2):87–89, March-April 2007.