Impacto de uso da abstração de software no desempenho de sistemas embarcados complexos

Vesmar Boris Camara C. [1], Ulisses B. Corrêa[1], Luigi Carro[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

{vbcamara, ubcorrea, carro}@inf.ufrgs.br

Abstract. *Nowadays major embedded systems functionalities are developed in software. Moreover, to attend the market exigencies the software productivity has to be improves. This work analyzes the overhead caused by the application of abstraction levels in embedded systems software development. This analysis was done based in Google Android. The obtained results showed that even about 80% of the executed instructions where in the lower levels from the layer architecture in applications that reuse framework components.*

Resumo. *Camadas de software servem para aumentar o nível de abstração e agilizam o projeto do sistema, por isso tem sido usados em sistemas embarcados. Este artigo analisa o uso de camadas de abstrações em software em sistemas embarcados complexos. Para isso, foram analisados dois cenários de uso de smartphones. Os resultados demonstram que camadas inferiores estão mais inter-relacionadas com o consumo (cerca de 80%) quando o desenvolvimento é fortemente baseado no reúso.*

1. Introdução

Os sistemas embarcados têm se tornado cada vez mais complexos, como pode-se ver nos *tablets*, reprodutores de mídia e *smartphomes* atuais. Assim, o projeto desses sistemas precisa deixar que novas funcionalidades possam ser definidas em etapas tardias, e até mesmo posteriores a comercialização do produto (customização pós-venda). Dessa forma, o caráter de comportamento destes sistemas deixou de ser homogêneo e o consumidor final é quem decide qual o conjunto de funcionalidades estarão disponíveis no produto adquirido.

Esse novo contexto criou uma série de desafios para os fabricantes de *smartphones*. Dentre esses desafios podemos citar: a necessidade do aumento da produtividade de software, o desenvolvimento de plataformas de hardware que sejam capazes de lidar com a diversidade de funcionalidades apresentando a eficiência requerida pelo domínio embarcado, entre outros.

A abstração é uma forma de entregar dispositivos ao mercado que permaneçam com um baixo custo e que satisfaçam a crescente demanda do mercado por novas funcionalidades (Purcell, 2007), por isso ela tem sido empregada frequentemente no desenvolvimento de ecossistemas computacionais complexos.

A engenharia de software propõe o emprego abstração para melhorar o processo de desenvolvimento de software, bem como, melhorar a qualidade do produto final

oferecendo uma quantidade considerável de opções de paradigmas, metodologias e técnicas para sua adoção. Isto posto, vale lembrar que 80% das funcionalidades atuais dos sistemas embarcados já vêm sendo desenvolvidas em software (Allan, 2008).

No entanto, a abstração de software pode ser a causa para diversos problemas em um sistema embarcado. Algumas vezes o uso de camadas de abstração é excessivamente utilizado por projetistas de software e, assim, impacta negativamente em fatores de qualidade do software embarcado, como o tamanho da memória cache utilizada para pilha de dados, TLB[1] *misses* e sobrecarga na chamada de funções (Smaalders, 2006).

O objetivo deste trabalho é identificar e medir a sobrecarga induzida pelo uso da abstração em sistemas de software para dispositivos embarcados, visto que esses são conhecidos por terem restrições rígidas quanto ao consumo de recursos físicos.

Como estudo de caso foi escolhida a plataforma de software para *smartphones* Android OS (Google Android, 2008). O impacto da abstração em termos de desempenho de execução será avaliado decompondo os consumos por camada de abstração desta plataforma, assim, possibilitando a análise de qual é o impacto real da abstração adotada por essa plataforma de software multicamadas no sistema embarcado.

Este trabalho esta organizado da seguinte forma. A Seção 2 apresenta o sistema operacional Android e seu framework. A Seção 3 mostra alguns trabalhos relacionados sobre a medição de desempenho. Após, a Seção 4 descreve a metodologia experimental. Na Seção 5 são discutidas algumas conclusões e trabalhos futuros.

2. Google Android Framework

O Google Android é uma pilha de software inicialmente desenvolvida para *smartphones*, mas que atualmente vem sendo empregado em diversos dispositivos embarcados, como *set top boxes*, *e-readers*, etc. Esta plataforma inclui um sistema operacional, um *midleware* e um framework de desenvolvimento que pode ser usado para desenvolver aplicativos em uma adaptação do framework Java.

2.1. Nível de Abstração de Software

Para o desenvolvedor de aplicações para Android, todas as camadas são abstraídas pela interface que é exposta a ele pela API do Framework de desenvolvimento.

Este Framework é livremente distribuído através do Android SDK o qual além de prover a API também inclui as ferramentas necessárias para o processo de desenvolvimento. O Framework possibilita o reúso ou extensão de seus componentes de forma facilitada.

Para tal, funcionalidades providas pelo sistema operacional e pelas bibliotecas são todas encapsuladas e expostas em uma camada adicional. Desta forma, todas as aplicações (camada topo da pilha) devem seguir o modelo de desenvolvimento que usa a abstração provida pelo nível imediatamente inferior. A Seção a seguir apresenta resumidamente cada uma das camadas do Android.

---

[1] *Translation Lookaside Buffer*.

2.2. Arquitetura do sistema

Android está constituído de cinco camadas de abstração de software (Google Android, 2008). Abaixo há uma breve descrição de cada delas, da camada base a camada topo.

-Linux Kernel: O Android é baseado no kernel do Linux 2.6, porém não é um "vanilla" Linux kernel, isto é, não faz parte do *branch* principal da árvore de desenvolvimento.

-Bibliotecas: Conjunto de bibliotecas C/C++ que são disponibilizadas ao desenvolvedor através do Framework.

-Runtime: Camada que abrange dois componentes essenciais para o funcionamento do sistema: a Dalvik VM (*virtual machine*) e as "*core libraries*" da linguagem Java.

-Framework: Criado para aumentar a produtividade, provendo reúso de componentes e abstraindo grande parte dos procedimentos necessários para a construção de uma aplicação.

-Aplicações: Camada onde estão os programas que interagem diretamente com o usuário.

3. Trabalhos relacionados

Existem vários estudos relacionados ao uso de abstração de software no contexto de sistemas embarcados. A maioria deles é sobre como a abstração de software e uso de camadas podem ajudar no desafio do projeto e desenvolvimento. No entanto, são poucos os trabalhos que se aprofundam na avaliação ou correlação da abstração com o desempenho do software produzido.

A abordagem de *Vertical Profiling* foi sugerida em (Hauswirth, 2004) com o propósito de enfrentar o desafio de entender como o consumo de ciclos de processador está disseminado nas diferentes camadas de software. Nesse trabalho, o autor incorporou o suporte a *profiling* vertical na Máquina Virtual Java (JVM) e demonstrou, através de cinco estudos de caso, que este método é efetivo para entender o comportamento das aplicações Java. Não obstante, este trabalho não relaciona os resultados com métricas importantes para sistemas embarcados, como dissipação de potência ou consumo energético.

Já em (Zhang, 2005), o cenário de dispositivos móveis é levado em conta, o autor demonstra que o desempenho não foi comprometido ao aplicar reúso de componentes de software. O estudo de caso foram quatro jogos RPG (*Role Playing Game*) diferentes, os quais foram analisados em termos de métricas de qualidade de software e desempenho.

Finalmente em (Batyuk *et al.*, 2009), os autores executaram um conjunto de *benchmarks* implementados de diferentes maneiras na plataforma Android. Os resultados mostraram que o uso do framework completo do Android pode diminuir o desempenho de um algoritmo de ordenação em até 30 vezes se comparado com a versão implementada em linguagem C do mesmo algoritmo e rodado na mesma plataforma Android. Aumentos no desempenho podem ser obtidos usando outras técnicas, tais como JNI que apresenta desempenho 10 vezes melhor que a implementação completamente focada no framework.

Este trabalho tem por objetivo mensurar a sobrecarga advinda da adoção de uma solução completa de abstração de software baseada em camadas e aplicada em sistemas embarcados. Trabalhos relacionados assumem que esta sobrecarga existe, outros medem a sobrecarga em cenários muito simples (pequenos algoritmos, conjunto de *benchmarks* simples, etc.). Este trabalho objetiva quantificar a sobrecarga que é adicionada na execução de aplicações reais do domínio móvel em sistemas onde a abstração de software é utilizada.

4. Experimentos

Os experimentos foram realizados em um ambiente montado para possibilitar a obtenção do desempenho de execução de aplicações (em termos de percentual de tempo de execução por camada) reais de *smartphones*.

Foram avaliadas várias ferramentas que podem proporcionar resultados almejados. Dentre elas foi escolhido o emulador de Android disponibilizado junto ao SDK oficial do Google. Conjuntamente, uma versão adaptada do *Traceview* é utilizada para a extração da informação de *profiling*, incluindo tempo de execução e o percentual relativo de tempo de execução para cada um dos métodos da aplicação inspecionada.

Um banco de dados com informações sobre as camadas de cada um dos métodos do Framework de desenvolvimento do Android foi criado, de forma semiautomática. Este banco contém os nomes dos métodos, sua classe e a qual camada ele pertence. Então, uma ferramenta foi criada para separar os consumos por camada, a partir da correlação do banco de informações sobre métodos e do *trace* de métodos executados.

Para estes experimentos, optou-se por utilizar aplicações web. Dois *benchmarks* de caráter diferente foram escolhidos: *Full web browser benchmark* e *User experience benchmark*. O primeiro utiliza o benchmark de navegadores Peacekeeper (Peacekeeper, 2011), enquanto o segundo simula tarefas cotidianas de usuários comuns (leitura de e-mails, operações na agenda, visualização de notícias e navegação).

A Tabela 1 apresenta os resultados obtidos depois de 5 execuções dos *benchmarks*. Os resultados demonstram uma concentração significativa do tempo de execução na Camada 2 (onde se encontram as bibliotecas nativas), isso se dá, pois, o navegador nativo do Android, utilizado nos experimentos, é fortemente baseado na biblioteca *libwebcore*, que possui implementação nativa e tem como objetivo disponibilizar uma *engine* de navegação reutilizável, sendo então exaustivamente testada pelo *benchmark*.

Tabela 1 Relação de tempo de execução por camada de Abstração.

Camada ou Nível	User Bench	Web Bench	Tempo (ms) Web Bench
camada 0: (indef.)	0,74%	8,02%	14534683
camada 2 (JNI):	54,27%	73,09%	132524680
camada 3:	14,45%	4,86%	8810053
camada 4:	30,42%	13,99%	25373279
camada 5:	0,13%	0,03%	62594

A ferramenta *traceview* gera resultados com base nos métodos Java executados sobre a Dalvik, impossibilitando avaliar os consumos dos métodos nativos da Camada 2 (*libraries*), ou da chamada do sistema operacional, Camada 1.

Para poder separar as Camadas 1 e 2 que estão catalogadas como JNI na Tabela 1, foi usada uma segunda abordagem, baseada na ferramenta de *profiling* do Oprofile (Levon e Elie, 2004) adaptada ao sistema Android, e os seus resultados são apresentados na Tabela 2. Esta tabela exibe as porcentagens de tempo de execução em que o sistema operacional (zImage) e as bibliotecas nativas executam.

Tabela 2 Resultados usando ferramenta de *profiling* do sistema para *Web Benchmark*.

%	%	Nome do código
93,81	Full Bench	
	51,2279	libwebcore.so
	18,5677	libc.so
	9,3496	libskia.so
	9,3	zImage (KERNEL)
	4,9117	libdvm.so
	1,2775	libm.so
6,0481	zImage (KERNEL)	
0,0814	oprofiled	

Faz-se interessante notar que o custo do mecanismo de instrumentação é bastante pequeno, justificando a sua escolha como método pouco intrusivo de obtenção dos dados almejados.

Tabela 3 Resultados usando ferramenta de *profiling* do sistema para *User Benchmark*.

%	%	Nome do código
88,03	User Bench	
	47,33	libwebcore.so
	15,42	zImage (KERNEL)
	9,00	libskia.so
	8,88	libc.so
	6,68	libdvm.so
	4,32	[heap] (tgid:67 range:0xa000-0x348000)
11,53	zImage (KERNEL)	
0,21	Oprofiled	

Da mesma forma, analisando a Tabela 3 nota-se que a *libwebcore* corresponde ao módulo que mais executou durante a operação do experimento. Outro binário que

chama a atenção é a *libdvm*, mas esta chama a atenção por sua baixa contribuição para o consumo da aplicação.

5. Conclusões e Trabalhos futuros

Os resultados experimentais apresentados demonstram que, para o sistema e os *benchmarks* utilizados, o impacto da aplicação de uma estrutura de camadas de abstração não pode ser categorizado negativamente, uma vez que, a maior parte do processamento se deu nas camadas mais baixas da estrutura de camadas de abstração. Isto nos mostra que o preço que pagamos para desenvolver novas aplicações de usuário, com novas funcionalidades feitas em software não é alto, pois menos de 1% do tempo de execução do código da camada 5 (aplicação) é realmente chamado, implicando um alto ganho de produtividade de software visto que há um alto nível de reúso de artefatos.

A partir dos resultados deste trabalho podem-se propor alguns trabalhos que avaliem mais a fundo determinados componentes do Android, como o custo de tradução da máquina virtual Dalvik. Outro trabalho que se pode propor é a investigação de meios para melhorar o desempenho nas camadas onde se encontra maior consumo de tempo de execução das aplicações. Isso pode ser feito através da aceleração de bibliotecas, ou do emprego de processadores dedicados para funções específicas. Ainda, no contexto dos experimentos aqui apresentados, cabe uma comparação com os mesmos *benchmarks*, porém utilizando-se outro navegador que não seja tão acoplado ao Android.

Referências

Allan, A., *et al.* (2002) "2001 Technology Roadmap for Semiconductors". IEEE Computer. January 2002.

Batyuk, L., *et al.* (2009) “Developing and Benchmarking Native Linux Applications on Android” in Mobile Wireless Middleware, Operating Systems, and Applications 2009.

Google Android, (2008) An Open Handset Alliance Project. Disponível em: <http://www.android.com>. Acessado em: 30/09/2011.

Hauswirth, M., Sweeney, P. F., Diwan, A., and Hind, M. (2004). “Vertical profiling: understanding the behavior of object-priented applications”. SIGPLAN Not.

Levon, J., Elie, P. Oprofile: A system profiler for linux. http://oprofile.sf.net, September 2004.

Peacekeeper (2011). Disponível em: <http://clients.futuremark.com/peacekeeper/index.action>. Acessado em: 30/09/2011.

Purcell (2007). Beyond Platformization: Using Mobile Software Management to Achieve Feature Customization of Mobile Phones.

Smaalders, B. (2006). Performance Anti-Patterns. Queue 4, 1 (February 2006), 44-50.

Zhang, W.; Jarzabek, S. (2005). Reuse without Compromising Performance: Industrial Experience from RPG SoftwareProduct Line for Mobile Devices. In: LNCS, n.3714, 2005.