Execução Remota de Aplicações para Computadores Móveis usando Aspectos

André Luiz Camargos Tavares, Marco Túlio de Oliveira Valente

Instituto de Informática
Pontifícia Universidade Católica de Minas Gerais

andrelct@gmail.com,mtov@pucminas.br

Abstract. *Software development for mobile computing devices requires new methods and tools that can handle the hardware and network limitations that are typical of these devices. In this paper, we describe a system called RDA (Remote Display with Aspects) that supports the execution of J2ME applications in server nodes located in fixed networks, while redirecting their presentation interfaces to mobile devices. The ultimate goal is to optimize the overall execution time of such applications. Aspects are used in RDA to intercept events related to interface design and update in J2ME systems. The paper also describes an experiment performed with our prototype implementation of RDA. This experiment shows that RDA can considerably optimize the performance of mobile applications based on time-consuming algorithms.*

Resumo. *Desenvolvimento de software para dispositivos computacionais móveis requer novos métodos e ferramentas que reduzam as restrições inerentes à natureza de tais dispositivos. Neste artigo, descreve-se o sistema RDA (Remote Display using Aspects), o qual viabiliza a execução de aplicações J2ME em servidores de uma rede fixa, mantendo a apresentação de suas interfaces em dispositivos computacionais móveis. O objetivo é reduzir o tempo total de execução de tais aplicações. Aspectos são intensivamente utilizados em RDA para prover interceptação dinâmica, transparente e não invasiva de eventos relacionados com a atualização de interfaces de sistemas J2ME. Experimentos realizados com RDA mostram que seu desempenho é consideravelmente superior à execução integral de certos sistemas em computadores móveis, notadamente naqueles casos em que estes sistemas demandam grande volume de processamento.*

1. Introdução

O crescimento da popularidade de dispositivos móveis, como PDAs e *smart phones*, tem demandado a investigação de novos métodos e ferramentas de desenvolvimento de *software*. Desenvolvedores de *software* para computadores móveis devem se preocupar com vários fatores: processamento limitado, capacidade de memória, tamanho do *display* e conectividade. Processamento limitado e capacidade de memória afetam a possibilidade de executar algoritmos complexos e manipular grandes quantidades de dados, dificultando a implementação de uma grande gama de aplicações e algoritmos em dispositivos móveis. Por outro lado, avanços constantes nas tecnologias de *hardware* têm beneficiado tanto servidores e estações de trabalho como computadores móveis. Assim, proporcionalmente, estima-se que servidores localizados em redes fixas continuarão ao longo dos anos

a possuir uma maior capacidade de processamento superior à disponível em computadores móveis. Por esta razão, uma maneira eficiente de executar aplicações para computadores móveis baseia-se na decomposição de um sistema em dois grupos de componentes: um para execução em servidores (com a lógica da aplicação) e outro contendo a interface do sistema, para execução em dispositivos computacionais móveis. Esta decomposição beneficia-se ainda de uma tendência de redes sem fio apresentarem larguras de banda cada vez mais próximas de tecnologias tradicionais de comunicação.

Embora diversos sistemas de programação suportem o desenvolvimento de aplicações distribuídas em ambientes cliente/servidor, eles freqüentemente exigem refatoramento não trivial do código, o que é particularmente complexo em ambientes caracterizados por recursos limitados. Para contornar este problema, este artigo apresenta um sistema de apoio à execução de aplicações para computadores móveis que realiza basicamente duas tarefas: (i) intercepta eventos de atualização de interfaces em aplicações Java desenvolvidas na plataforma J2ME [9] e executadas em servidores localizado em uma rede fixa e (ii) direciona estes mesmos eventos para um dispositivo computacional móvel.

O sistema proposto, chamado RDA (*Remote Display using Aspects*), decompõe dinamicamente uma aplicação em dois grupos de componentes: lógica e interface gráfica. Para que a decomposição seja feita transparentemente e sem nenhuma alteração na aplicação, são usados aspectos, implementados em AspectJ [6], que capturam eventos que alteram o *display* da aplicação. Assim, a lógica da aplicação é integralmente executada no servidor, enquanto a interface gráfica é redirecionada para o cliente móvel. Assim em RDA, o uso de aspectos assegura que a decomposição de aplicações seja feita de forma dinâmica, não invasiva e transparente.

O restante deste artigo está organizado conforme descrito a seguir. A Seção 2 apresenta o funcionamento básico do sistema proposto. A Seção 3 descreve os aspectos propostos em RDA para interceptação e redirecionamento de eventos de atualização de interface. A Seção 4 apresenta os resultados de um experimento realizado com a atual implementação do sistema. Na Seção 5 são discutidos trabalhos relacionados. Finalmente, a Seção 6 conclui o artigo e relaciona trabalhos futuros.

2. Funcionamento do Sistema Proposto

O sistema RDA inclui um servidor de execução remota e um conjunto de dispositivos computacionais móveis, incluindo PDAs e celulares. O servidor de execução remota é responsável por executar aplicações e redirecionar a interface das mesmas para dispositivos computacionais móveis. Inicialmente, as aplicações móveis sujeitas a execução remota devem ser transferidas para este servidor, conforme mostrado na Figura 1. Para isso, o usuário de um dispositivo móvel deve se conectar ao servidor (passo 1 da Figura 1) e informar o nome dos arquivos que formam a aplicação. Estes arquivos serão então transferidos para o servidor (passo 2). Em seguida, o sistema dispara um processo de *weaving*, por meio do qual são incorporados aos arquivos transferidos aspectos responsáveis por redirecionar a interface da aplicação para dispositivos móveis. Concluído o processo de *weaving*, a aplicação está apta a ser executada no servidor de execução remota, com sua interface sendo redirecionada para um dispositivo móvel (incluindo não apenas o dispositivo mencionado no passo 1, mas também computadores móveis de outros usuários do sistema RDA).

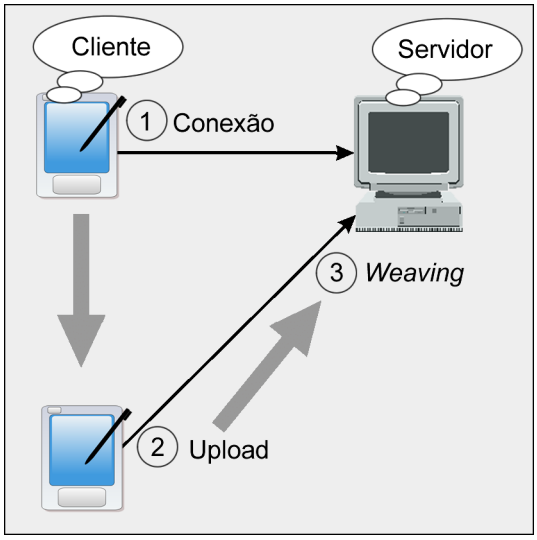

Figura 1. Transferência de Aplicações para o Servidor do Sistema RDA

A fim de executar uma aplicação previamente transferida para o servidor de execução remota, o usuário de um computador móvel deve se conectar a este servidor (passo 1 da Figura 2), que lhe envia a lista de aplicações disponíveis para execução (passo 2). O usuário seleciona a aplicação e solicita sua execução (passo 3). A partir deste momento, a execução desta aplicação inicia-se no servidor (passo 4), com sua interface sendo automaticamente redirecionada para o dispositivo móvel do usuário.

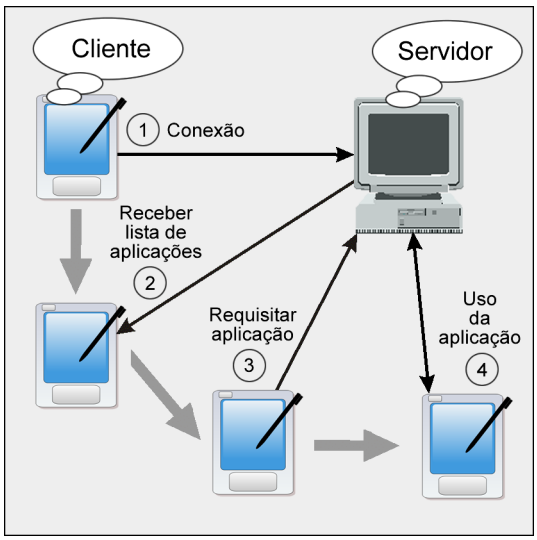

Figura 2. Execução de Aplicações no Sistema RDA

3. Aspectos Propostos

A Figura 3 mostra a arquitetura de execução de uma aplicação usando RDA. A primeira camada do lado do servidor (lcduiServidor) representa a aplicação que será executada no servidor e terá a interface gráfica mostrada no cliente. A camada da aplicação é executada no servidor usando um micro-emulador de J2ME [5] . Este micro-emulador é executado em J2SE no servidor e emula aplicações desenvolvidas originalmente para J2ME.

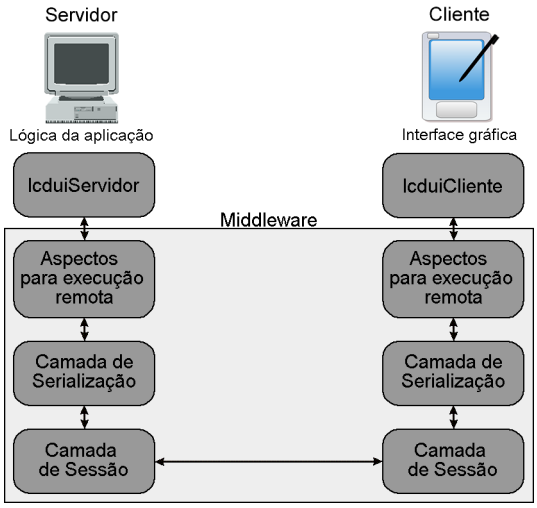

Figura 3. Arquitetura da execução

Quando a aplicação invoca um método do pacote `lcdui` de J2ME (pacote contendo métodos para desenvolvimento de interfaces gráficas), aspectos são usados para capturar estas invocações e redirecioná-las para o cliente. A camada de serialização implementada em RDA trabalha com a serialização e deserialização das informações transferidas entre o servidor e o cliente móvel. A camada de sessão lida com a criação e manutenção da conexão cliente-servidor e a transferência de objetos e métodos serializados. A atual implementação de RDA utiliza um protocolo próprio de comunicação, baseado em *sockets*.

Objetos de Lógica e de Interface: Em J2ME, uma aplicação pode criar um ou mais objetos gráficos. Os objetos gráficos criados no servidor são interceptados por um aspecto e um objeto idêntico é criado no cliente. Estes objetos são assim denominados no sistema RDA:

- **Objeto de Lógica (OL):** objeto associado a um componente gráfico (como um botão, área de texto, painel etc). Este objeto é acessado pela lógica da aplicação e, portanto, instanciado no servidor do sistema RDA.

- **Objeto de Interface (OI):** objeto idêntico ao OL, porém instanciado em um dispositivo computacional móvel.

Todo OL possui uma referência de rede para seu respectivo OI. Esta duplicidade de objetos existe para que RDA posssa garantir a consistência da aplicação. A existência de um OI no cliente associado a seu correspondente OL no servidor garante que uma chamada de método realizada sobre o OL possa ser propagada para o cliente, de forma a atualizar a interface do mesmo.

A camada de aspectos para execução remota de aplicações móveis em servidores contém todos os aspectos de RDA. Quando um método ou um construtor de interface gráfica é invocado no servidor, o sistema RDA intercepta esta invocação. Após esta interceptação, são iniciadas duas *threads*. A primeira continua a execução normal do método ou construtor no servidor, para que os seus OL sejam atualizados. Porém, não

se exibe nenhuma informação no servidor, pois o método de inicialização da interface gráfica é capturado e impedido de continuar sua execução normal. Já a segunda *thread* serializa a ação capturada e envia seus dados para o cliente. Uma vez recebidos no cliente, os dados são deserializados e o método ou construtor é executado no respectivo OI do cliente.

A camada de aspectos para *display* remoto de interfaces em clientes móveis contém também *listeners*, os quais são responsáveis por capturar as entradas de dados do usuário. Uma vez que um *listener* recebe uma entrada de dados do usuário (por exemplo, o usuário clicou em um botão da interface), gera-se uma ação para serializar esta entrada e, então envia-se para o servidor os dados serializados. No servidor, estes dados são deserializados e executa-se o respectivo *listener* do OL.

Descreve-se a seguir os demais aspectos empregados na implementação de RDA:

Aspectos para interceptar o início da aplicação: A fim de viabilizar esta interceptação, o seguinte *pointcut* é usado:

```
pointcut MIDletInicialization():
    staticinitialization (javax.microedition.midlet.MIDlet+);
```

Este *pointcut* intercepta a inicialização estática da classe da aplicação que estende a classe `MIDlet` de Java. Um *advice* do tipo `before` associado a este *pointcut* inicia então uma conexão do servidor com o cliente. Em RDA, usa-se *sockets* para implementar esta conexão.

Aspectos para interceptar a inicialização da interface gráfica: O método `setCurrent` da classe `Display` é o único responsável por exibir uma tela de interface em aplicações J2ME. A seguir, mostra-se o *pointcut* usado para interceptar execuções deste método.

```
pointcut setCurrent(Displayable nextDisplayable,Display display):
    call(void Display.setCurrent(Displayable)) &&
    args(nextDisplayable) && target(display);
```

Um *advice* do tipo `around` é associado a este *pointcut* com o objetivo de desabilitar a execução do método `setCurrent` no servidor. Assim, nenhuma informação de interface será exibida neste servidor. Em vez disso, os dados deste método são propagados para o cliente para que sua execução seja devidamente disparada no dispositivo móvel do usuário.

Aspectos para interceptar instanciações de objetos de interface gráfica: Essa interceptação é feita nos construtores dos objetos do pacote `javax.microedition.lcdui` de J2ME, por meio de diversos *pointcuts*. A seguir, segue um exemplo da interceptação da instanciação de um objeto da classe `Form`:

```
pointcut FormConstructor1(String title):
   call(Form.new(String)) && args(title);
```

Um *advice* do tipo `after returning` associado a este *pointcut* captura o objeto OL instanciado no servidor e armazena seu *hashCode* em uma tabela. Em seguida, são enviados para o cliente os seguintes dados do objeto criado: nome de sua classe (por exemplo, `Form`), nome do construtor invocado (por exemplo, `FormConstructor1`) e o valor de seus argumentos. Quando estas informações são recebidas pelo cliente, um respectivo objeto OI é criado e armazenado em uma tabela na mesma posição que o *hashCode* de seu correspondente OL no servidor.

Além do *pointcut* mostrado, foram implementados outros quinze *pointcuts* para interceptação de instanciações de outros tipos de componentes de uma interface J2ME.

Aspectos para interceptar invocações de métodos de objetos de interface gráfica: Essa interceptação é feita sobre os métodos do pacote `javax.microedition.lcdui` de J2ME, por meio de diversos *pointcuts*. A seguir, segue um exemplo de interceptação do método `setString` de um objeto da classe `TextField`:

```
pointcut SetString(String text, TextField textField) :
    call(void TextField.setString(String)) &&
    args(text) && target(textField);
```

Um *advice* do tipo `before` associado a este *pointcut* captura as informações do método. Antes do método ser executado no servidor, envia-se para o cliente as seguintes informações sobre o mesmo: nome da classe à qual o método pertence (por exemplo, `TextField`), nome do método invocado (por exemplo, `setString`), valores dos argumentos e o índice do *hashcode* do OL sobre o qual o método foi invocado, na tabela de *hashCode* de objetos. Quando estas informações são recebidas pelo cliente, o método `setString` da classe `TextField` é invocado no respectivo OI com os mesmos argumentos da chamada realizado no servidor. Caso um dos argumentos seja um OL, envia-se para o cliente a posição deste objeto na tabela de *hashcodes*. Após a execução do *advice*, o método é executado normalmente no servidor para manter o OL atualizado, porém sem gerar nenhuma resposta gráfica. Desta maneira, métodos que apenas retornam alguma informação de um objeto, como o método `getLabel`, não precisam ser interceptados, pois o próprio OL possui as informações que devem ser retornadas, sem que seja preciso consultar o OI do cliente.

Além do *pointcut* mostrado, foram implementados outros quarenta e oito *pointcuts* para interceptação de invocações de métodos sobre outros tipos de componentes de uma interface J2ME.

4. Resultados Experimentais

A fim de testar o desempenho do sistema RDA, foi montado um experimento incluindo um Palm (cliente) conectado em rede com um servidor por meio de uma rede IEEE 802.11. A aplicação de exemplo usada no experimento faz uso do Algoritmo de Dijkstra para calcular o menor caminho entre dois vértices de um grafo. Este tipo de *software* pode ser usado, por exemplo, para se buscar o melhor caminho entre dois pontos de uma cidade.

A aplicação foi executada em grafos com 10, 150, 500 e 1500 vértices. Além disso, para cada um destes grafos, variou-se o percentual de arestas do grafo em relação

a um grafo completo com o mesmo número de vértices. As máquinas usadas no experimento foram as seguintes: Palm Lifedrive, Intel Xscale 416MHz, 64Mb de RAM, IBM JVM J9 2.2 e um AMD Athlon64 1.80GHz, 1024Mb de RAM, Windows XP Professional SP2.

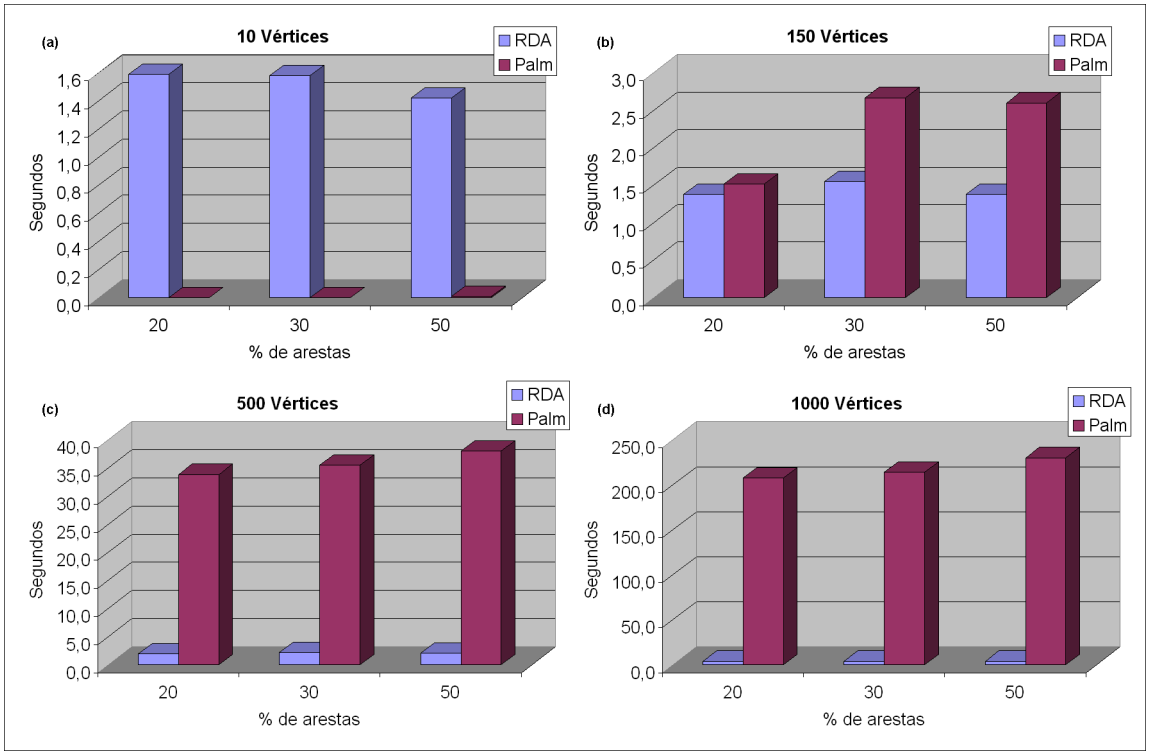

Figura 4. Resultado do experimento realizado

A Figura 4 compara o tempo de execução da aplicação usando os sistema RDA e o tempo de execução quando a aplicação foi integralmente executada no Palm. No primeiro experimento (Figura 4a), o resultado do sistema RDA não foi compensador, tendo em vista que o grafo usado possuía apenas 10 vértices, o que acabou por demandar um volume de processamento pequeno o suficiente para não compensar o *overhead* de comunicação inerente ao uso do sistema RDA. Porém, com o aumento da quantidade de vértices, pode-se notar que eficiência do emprego de RDA aumenta consideravelmente. No segundo experimento, com 150 vértices, o tempo de execução via RDA se manteve semelhante ao do primeiro experimento, enquanto o tempo da execução direta no Palm subiu para 3 segundos. Nos dois experimentos seguintes, o tempo de execução via RDA aumentou um segundo apenas, de um experimento para o outro. Por outro lado, o tempo da execução direta no Palm aumentou para 40 segundos no terceiro experimento, e posteriormente para quase 250 segundos no último experimento.

Esta avaliação mostra que, quando se trata de aplicações com grandes cargas de processamento, o uso do sistema RDA contribui para reduzir consideravelmente o tempo total de execução.

5. Trabalhos Relacionados

Diversos sistemas para execução remota de aplicações já foram propostos. Dentre eles, podemos citar os seguintes: TCPTE [3, 2], ULC [1], X Window [7] e RDP [4].

Thin Client Applications for Limited Devices (TCPTE) [3, 2] é um *framework* para o desenvolvimento de aplicações Java-AWT para dispositivos móveis. TCPTE suporta migração de aplicações J2SE para dispositivos com recursos limitados através da redução do processamento do lado do cliente. O sistema usa uma arquitetura de apresentação remota semelhante à adotada em RDA, com objetos de lógica (do lado do servidor) e de interface (do lado do cliente). Utiliza-se ainda um protocolo de comunicação otimizado para redes sem fio. *Ultra Light Client (ULC)* [1] é outro sistema de apresentação remota baseado no conceito de objetos de lógica e de interface.

Sistemas de apresentação remota já foram propostos também para o gerenciador de janelas X-Window [7], comum em ambientes Unix, e para sistemas Microsoft Windows, por meio do sistema Remote Desktop Protocol (RDP) [4]. VMware é um outro sistema largamente utilizado para criação de máquinas virtuais que emulam processadores Intel x86 [10]. Essencialmente, o sistema permite que múltiplos sistemas operacionais executem em uma mesma estação (permitindo, por exemplo, que aplicações Windows sejam executadas em máquinas cujo sistema operacional nativo é Linux). De forma semelhante a RDA, o sistema permite ainda redirecionar o *display* de uma máquina virtual para uma estação remota.

O padrão de projeto conhecido como *half-objects* [8] documenta os problemas e desafios inerentes à manutenção de cópias de um mesmo objeto em espaços de endereçamento distintos. Em essência, este padrão norteou o projeto dos objetos de lógica e de interface usados em RDA.

6. Conclusões

Neste artigo, apresentou-se um sistema orientado por aspectos para a execução remota de aplicações J2ME. Para que uma aplicação seja executada remotamente, sua interface gráfica deve ser enviada para o dispositivo cliente. Para isso, o sistema RDA usa de forma intensiva aspectos que interceptam invocações de métodos da interface gráfica. Assim, o uso de aspectos possibilita que a execução remota seja feita sem necessidade de alteração da aplicação alvo.

Em RDA, clientes equipados com dispositivos móveis conectam-se ao servidor do sistema e transferem suas aplicações para os mesmos, as quais ficarão disponíveis para que outros usuários possam usá-las. Portanto, o servidor do sistema pode se tornar um repositório de aplicações passíveis de execução remota.

A atual implementação do sistema é compatível com aplicações desenvolvidas usando MIDP 1.0. No entanto, estas aplicações não devem usar a classe `Canvas` do pacote `javax.microedition.lcdui`, responsável pela definição de gráficos em formato *bitmap*. Como trabalho futuro, pretende-se estender o sistema para atender ao perfil MIDP 2.0 e investigar estratégias para minimizar o custo de comunicação inerente ao uso de RDA (por exemplo, agrupando mensagens trocadas entre computadores móveis e fixos). Além disso, pretende-se realizar outros experimentos com o sistema, envolvendo aplicações comuns no ambiente J2ME.

Agradecimentos: Este trabalho foi desenvolvido como parte de um projeto de pesquisa financiado pela FAPEMIG (processo CEX-817/05 - Edital Universal 2005).

Referências

[1] Canoo Engineering AG. Ultra Light Client (ULC): Technology White Paper. http://www.canoo.com/ulc/home/whitepaper.html.

[2] Gerardo Canfora, Giuseppe Di Santo, and Eugenio Zimeo. Developing Java-AWT thin-client applications for limited devices. *IEEE Internet Computing*, 9(5):55–63, 2005.

[3] Gerardo Canfora, Giuseppe Di Santo, and Eugenio Zimeo. Developing and executing Java AWT applications on limited devices with TCPTE. In *28th International Conference on Software Engineering*, pages 787–790. ACM Press, 2006.

[4] Microsoft Corporation. Remote Desktop Protocol (RDP) Features and Performance. http://support.microsoft.com/kb/186607.

[5] J2ME Micro Emulator. http://sourceforge.net/projects/microemulator/.

[6] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of AspectJ. In *15th European Conference on Object-Oriented Programming*, volume 2072 of *LNCS*, pages 327–355. Springer Verlag, 2001.

[7] N. Masnfield. *The Joy of X: An Overview of the X Window System*. Addison Wesley, 1993.

[8] Gerard Meszaros. Pattern: half-object + protocol (HOPP). *Pattern languages of program design*, pages 129–132, 1995.

[9] Roger Riggs, Antero Taivalsaari, and Mark VandenBrink. *Programming Wireless Devices with the Java 2 Platform, Micro Edition*. Addison Wesley, 1th edition, July 2001.

[10] VMware. http://sourceforge.net/projects/microemulator/.