Ambiente Java para sistemas integrados de recursos limitados: Aplicação num sistema da JUMPtec

Valter Filipe Silva[1], José Alberto Fonseca[2], José Luís Oliveira[2]

1 – Escola Superior de Tecnologia de Castelo Branco, Castelo Branco, vfs@est.ipcb.pt
2 – Departamento de Electrónica e Telecomunicações, Universidade de Aveiro {jaf,jlo}@det.ua.pt

Os sistemas de controlo e automação são normalmente baseados em sistemas integrados com poucos recursos. No entanto, mesmo com estas limitações por vezes é necessário que contenham ligação à Internet e rápidos processos de desenvolvimento. Este artigo apresenta um demonstrador baseado num sistema integrado da JUMPtec por forma a exibir a possibilidade do uso da linguagem de programação Java em pequenos sistemas de recursos limitados. Os passos da integração da máquina virtual e a simplicidade de desenvolvimento de aplicações é demonstrada. São também apresentados alguns resultados comparativos e alguns exemplos de aplicação.

INTRODUÇÃO

Nos nossos dias quem desenvolve aplicações deseja obter resultados eficientemente e rapidamente. Os programadores também desejam utilizar linguagem com as quais estejam familiarizados. Para computadores de secretária e servidores muitos programadores escolhem o Java [2] devido às funcionalidades que este oferece. Algumas destas funcionalidades são o *Garbage Collector*, independência de plataforma e a sua similaridade com o C++.

Nos sistemas integrados, as linguagens de programação normalmente utilizadas são o C e o *Assembly*. O Assembly está cada vez mais em desuso, excepto para pequenos procedimentos que necessitam de ser rápidos ou para aceder a *Hardware* especifico. Portanto o normal é ser usado o C. Mesmo assim, programar sistemas integrados em C, pode ser uma tarefa difícil caso não seja feito o devido encapsulamento e uma correcta da memória.

Quando se considera o uso do *Java* em sistemas de recursos limitados a ideia que este é lento e ocupa muita memória ainda persiste. No entanto nos nossos dias é possível correr programas escritos em *Java* em sistemas com recursos limitados quando comparados com os computadores de secretária., tais como telefones móveis[10] ou outros sistemas de computação móveis. A diversidade deste tipo de sistemas é muita (por exemplo nos telefones móveis) e logo o *Java* com a sua independência de plataforma pode ser uma boa solução para a programação rápida destes sistemas.

É importante referir que o alvo das primeiras versões do *Java* era precisamente estes sistemas de electrónica[2] de consumo, mas com o crescimento do sistema *Java* este objectivo foi abandonado. Hoje em dia está novamente a surgir o interesse em mover o *Java* para os sistemas de recursos limitados para trazer para este mundo as funcionalidades oferecidas para os computadores de secretária.

Neste artigo, o problema de portar o *Java* para um sistema integrado de recursos limitados é discutido. Neste caso particular o sistema usado o sistema alvo é um sistema usado em aplicações de controlo e automação. A plataforma escolhida foi um sistema da JUMPtec que se pode considerar um sistema com alguns recursos para o sector em que se insere.

O sistema integrado da JUMPtec

Neste trabalho usou-se um JUMPtec WebToNet (figura 1). Este é baseado num processador 386SX com 2Mb de memória RAM e 2Mb de memória *flash*, porta série, porta paralela e placa de rede[1]. Esta plataforma foi escolhida devido à sua boa relação preço desempenho. A figura 1 apresenta duas vistas do sistema. As suas dimensões físicas são 11,2x11,6cm.

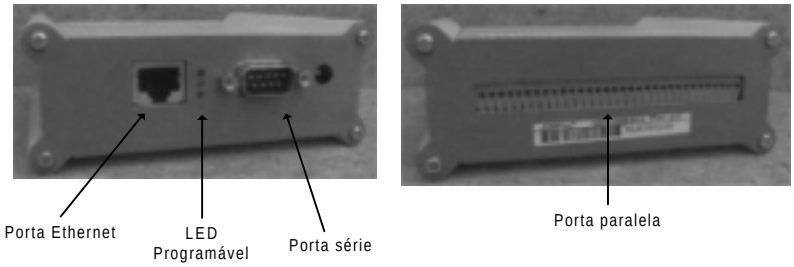

Figura 1. O sistema usado

Este sistema integra um servidor *Web* que permite que se gere e controle o sistema a partir de qualquer *Web Browser*. Em particular, este é muito útil para a gestão do disco *flash* (figura 2). Este sistema tem atribuído um endereço IP e um nome como qualquer outro sistema numa rede. O servidor *Web* é executado simultaneamente com o sistema operativo, o que permite manter a comunicação com o sistema enquanto se executa outra aplicação.

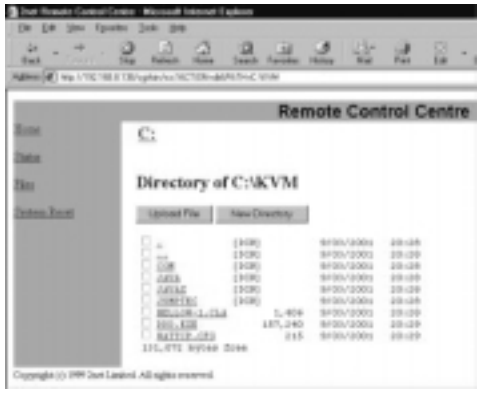

Figura 2. O interface de gestão do disco *flash*.

Este sistema tem o Dr-DOS como sistema operativo, que é compatível com o Ms-DOS, sendo mais orientado para sistema de recursos limitados. Assim todos os programas desenvolvidos para o sistema podem ser previamente testados num sistema de secretária com este sistema operativo.

Integração da máquina virtual *Java* num sistema limitado

Nos nossos dias existem três edições do *Java*, cada uma orientada para um mercado:

- *Java 2 Enterprise Edition* (J2EE) – Para o grandes servidores oferecendo uma solução de Internet sólida e escalonável.
- *Java 2 Standard Edition* (J2SE) – Para os computadores de secretárias e as suas aplicações comuns.
- *Java 2 Micro Edition* (J2ME) – Esta edição é dedicada aos pequenos dispositivos com fraca capacidade de processamento e pouca memória.

Java 2 Micro Edition é uma edição muito flexível e portável sendo esta funcionalidades providenciadas na sua arquitectura baseada em configurações e perfis. Uma configuração define as funcionalidades básicas para uma família de dispositivos com capacidades similares. Um perfil adiciona funcionalidades especificas a uma configuração particular.

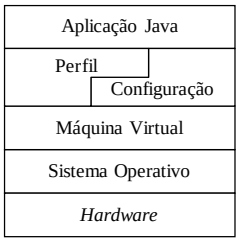

Figura 3. A arquitectura do J2ME

Presentemente existem 2 configurações suportadas pelo J2ME: CLDC *Connected Limited Device Configuration* e CDC *Connected Device Configuration*. O CDC usa a máquina virtual clássica enquanto que o CLDC usa a nova *Kilo Virtual Machine* (conhecida por KVM).A configuração CLDC está disponível no site da *Sun* [3] para *download* grátis. O conjunto que se obtém através do site da *Sun* inclui a máquina virtual, as classes necessárias para a execução de aplicações e algumas ferramentas, como o *preverifier*, tudo escrito em C e em *open source*.

O sistema operativo presente no JUMPtec é DOS, portanto é necessário portar o KVM para este sistema operativo. De acordo com o guia de portabilidade do KVM [5], o código tem de ser compilado com um compilador ANSI C com ponteiros 32 bits. Assim usou-se o Gnu C [6] para compilar o KVM. Este compilador suporta aritmética de 64 bits, mas para que seja mantida a compatibilidade com outros compiladores alguns funções têm de ser providenciadas. Isto é necessário porque o tipo de dados `long` é de 64 *bits*. Estas funções estão identificadas na tabela 1.

Tabela 1. Funções desenvolvidas

Função	**Equivalente *Java***
`long64 ll_mul(long64 a,long64 b);`	`a*b`
`long64 ll_div(long64 a,long64 b);`	`a/b`
`long64 ll_rem(long64 a,long64 b);`	`a%b`
`long64 ll_shl(long64 a,long64 b);`	`a<<b`
`long64 ll_shr(long64 a,long64 b);`	`a>>b`
`long64 ll_ushr(long64 a, long64 b);`	`a>>>b`

O tipo de dados `long64` é uma estrutura com dois inteiros.

Muitas das mais populares aplicações para sistemas integrados são os sistemas de controlo, que normalmente usam aritmética de virgula flutuante. Assim, foi incluído na máquina virtual o suporte para este tipo de aritmética. As funções necessárias são apresentadas na tabela 2.

Tabela 2. Funções para suporte de virgula flutuante

Função	Equivalente Java
`long64 float2ll (float f);`	`(long)f`
`long64 double2ll (double d);`	`(long)d`
`float ll2float (long64 a);`	`(float)a`
`dlouble ll2double (long64 a);`	`(dlouble)a`

O nosso processador alvo é um 386Sx, e logo ele não tem unidade de ponto flutuante. Desta forma foi necessário emular o ponto flutuante em *software* juntando para tal a biblioteca que providencia estas funcionalidades. Outras funções foram desenvolvidos para aceder aos portos de *hardware* de uma forma controlada e para fazer tarefas mais especificas, tais como algumas tarefas no *Class Loader*.

Outro problema no portar do KVM para DOS é o tamanho dos nomes dos ficheiros, que apenas são de 8 caracteres mais 3 de extensão. Em *Java*, os ficheiros compilados têm uma extensão de "`.class`" e o nome das classes podem conter mais de 8 caracteres (formato *Joliet*). Uma das possibilidades, seria limitar o tamanho o tamanho das classes *Java*. Isto não seria muito apreciado pelos programadores e iria contra as especificações *Java*. Assim decidiu-se alterar o *Class Loader* para superar esta limitação. Este problema foi resolvido porque o ficheiro "`.class`" contém o seu nome na sua estrutura interna [7]. Assim, antes de carregar alguma classe o sistema verifica se o nome da classe contém mais de 8 caracteres. Se não deve existir no disco um ficheiro exactamente com o mesmo nome. Se o nome é maior que 8 caracteres, o sistema gera uma sequência de nomes baseada nos 6 primeiros caracteres adicionando os caracteres ~#. Onde # é um número de 1 a 9. O sistema abre cada ficheiro até encontrar o pretendido. Um exemplo deste tipo de nomes é apresentado na tabela 3.

Tabela 3. Exemplo de duas classes e respectivos novos

Nome em formato Joliet	Nome DOS
my_program.class	my_pro~1.cla
my_program1.class	my_pro~2.cla

O novo algoritmo usado na modificação do *Class Loader* foi obtido do original e é apresentado na figura 4.

Todas as aplicações que sejam executadas no KVM têm de ser pré-verificadas [8]. Por causa do problema do tamanho dos ficheiros, o pré-verificador foi também alterado para que a extensão dos ficheiros seja alterado de "`.class`" para "`.cla`".

Poder-se-ia usar o tipo de ficheiros JAR para resolver esta limitação. Esta aproximação levaria a uma máquina virtual incapaz de executar uma aplicação num único ficheiro, e logo não de acordo com as especificações *Java*. Por outro lado o utilitário de pré-verificação não suporta ficheiros JAR e logo todas as

classes teriam que ser pré-verificadas antes de serem incluídas neste ficheiro JAR. Também não seria possível dar nomes longos aos ficheiros JAR.

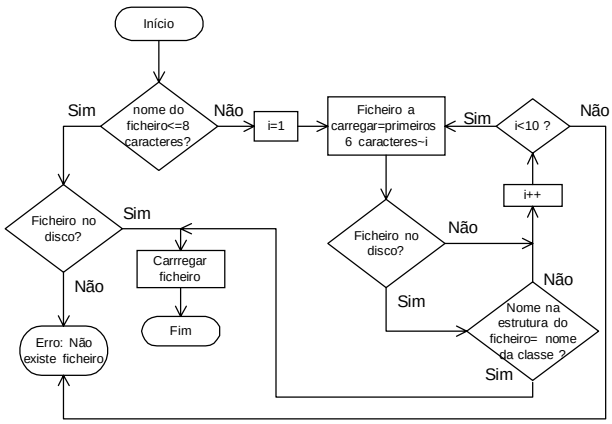

Figura 4. Algoritmo do *Class Loader*

Em *Java* é também possível de definir funções nativas, isto é funções especificas para uma determinada plataforma. Como é apresentado na figura 5 isto permite ao programador criar aplicações mais especificas. No caso do KVM todas as funções nativas têm que ser compiladas juntamente com o KVM, o que não é o procedimento usual nas restantes máquinas virtuais.

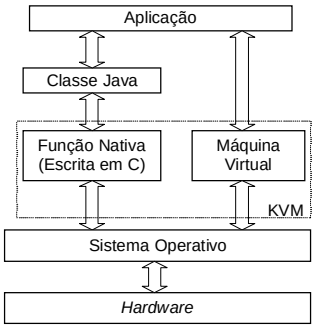

Figura 5. Funções nativas

Para o JUMPtec duas funções nativas foram desenvolvidas para existir acesso aos portos de *hardware*. Estas funções são úteis não só para *debug*, mas também para aplicações típicas deste sistema. O KVM foi também modificado para redireccionar todos os avisos para a porta série.

Para se ter acesso à placa *Ethernet* em DOS, o JUMPtec disponibiliza um *packet driver*. Neste caso para existir acesso à rede a partir do *Java* decidiu-se usar a *Connection Framework* disponível com o CLDC (parcialmente implementada com funções nativas) junto com uma biblioteca de funções obtida como parte do sistema *wattcp*. O sistema *wattcp* providencia o acesso a uma placa de rede através de um *packet driver*. Este *software* é grátis e está disponível em *open source* em [9].

Desenvolvimento de programas

Na figura 6, os passos necessários para o desenvolvimento de aplicações em *Java* para o JUMPtec são apresentados. O primeiro passo é, obviamente a escrita da aplicação num editor de texto comum num PC de

secretária. Todas as sintaxes usadas devem ser exactamente as mesmas que para qualquer outro sistema. O programa deve ser compilado com o `javac` existente no *Java Development Kit* (JDK).

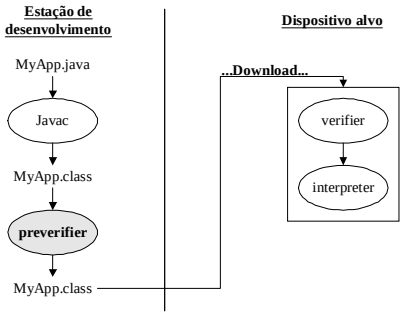

Figura 6. Desenvolvimento de programas

Todas as máquinas virtuais devem ser capazes de identificar e rejeitar classes inválidas. No entanto, porque o processo de verificação existente no J2SE usa muitos recursos, este não é passível de ser usado em dispositivos de recursos limitados. Assim, o CLDC define uma forma alternativa de para a verificação das classes em que um atributo é adicionado a cada método numa operação feita no dispositivo de desenvolvimento, chamada de pré-verificação. Esta operação é especifica do método de desenvolvimento de programas para a KVM. No entanto as classes modificadas por este processo continuam a poder ser executadas em qualquer outra máquina virtual [8], tendo um incrementa no tamanho de cerca de 5%.

Depois desta pré-verificação a classes está pronta para ser transferida para o JUMPtec. Esta operação pode ser feita com recurso à aplicação apresentada na figura 2.

Para deixar para o KVM o máximo de memória possível, um pequeno programa de arranque foi desenvolvido. Este programa foi escrito em C e consoante o bit 0 da porta paralela executa o KVM ou o servidor Web. Desta forma a ausência de teclado é minimizada.

Antes de executar a máquina virtual o *Packet Driver* tem que ser instalado para existir acesso à placa *Ethernet* a partir do programa *Java*. Pela mesma razão é necessário existir um ficheiro presente no directório do KVM chamado `wattcp.cfg` [9] que contém as configurações de rede, como o endereço IP, a porta de saída ou o domínio. De notar que o programa de arranque apenas é executado no arranque do sistema, e logo sempre que se desejar executar o nosso programa deve-se reinicializar o sistema. Esta operação pode ser feita com a aplicação da figura 2.

Para facilitar o desenvolvimento de aplicações duas *packages* foram desenvolvidas.

Package `jumptec`

Esta *package* tem três importantes classes, a classe *Port*, a classe *Net* e a classe *Led*. A classe *Net* tem alguns métodos derivados do *wattcp*[9]. A sua funções é providenciar algumas funcionalidades básicas de rede.

A classe Port tem dois métodos estáticos semelhantes ao `outportb()` e `inportb()` de C.

A classe Led serve para ligar e desligar o *led* localizado na parte da frente do JUMPtec (figura 1). É útil para diagnóstico e *debug*.

Package `jumptec.comm`

Este *package* tem duas importantes classes, uma para gerir as comunicações da porta série e outra para gerir as comunicações da porta paralela. Algumas funcionalidades clássicas são fornecidas para estes dois tipos de interface.

Exemplos de aplicação

Para que a aplicação ser executada é necessário transferi-la para o JUMPtec e pelo menos as classes necessárias à sua execução. A KVM ocupa 325Kb, e se forem instaladas todas as classes são necessários mais 525 Kb, portanto o sistema completa ocupa 850Kb. No entanto o JUMPtec usa o disco *flash* dividido em blocos de 4Kb, logo um ficheiro ocupa um espaço múltiplo de 4Kb, o que leva a algum espaço desperdiçado.

Dois básicos exemplos foram preparados para demonstrar a funcionalidade do sistema. O primeiro é o uso numa típica aplicação de controlo, enquanto que o segundo demonstra o funcionamento do sistema com a placa de rede.

Todas as aplicações foram desenvolvidas num PC de secretária e podem ser testadas usando o DOS ou uma janela de DOS no sistema operativo Windows. Isto é útil para teste e é possível porque o JUMPtec é um sistema compatível com o sistema IBM PC.

O cenário de aplicação do primeiro exemplo é apresentado na figura 7. Neste exemplo, dependendo do caracter enviado pelo PC, os *leds* correspondentes ligam ou desligam. O estado dos *Switches* também podem ser lido. Esta aplicação é muito simples, mas demostra as funcionalidades do sistema para interacção com o mundo exterior necessário em muitas aplicações deste tipo de sistemas.

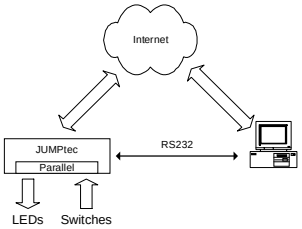

Figura 7. Exemplo de aplicação

Com o *Java* o tempo de desenvolvimento das aplicações é reduzido quando comparado com outras linguagens. Além desta vantagem, é de notar que tamanho desta aplicação é de cerca de 1Kb. Nesta experiência o tempo de arranque foi de cerca de 5 segundos, incluindo o arranque da KVM, do DOS e do *reset* do sistema.

Para avaliar o desempenho do *Java* quando comparado com outras linguagens, o tempo de um ciclo do *led* a ligar e desligar foi medido. Os resultados são apresentados na tabela 4. Como esperado em *Java* o tempo é maior numa ordem de grandeza. No entanto o resultado também mostra que a resposta do sistema é adequada para muitas aplicações de controlo e instrumentação. De facto se o processo não envolver movimento (como robôs), muitos processos industriais, como controlo da temperatura podem aceitar tempos muito superior muito maiores. Se a comparação for feita especificamente entre o *Java* e o C++, deve-se notar que o tamanho do código transferido é muito menor quando se usa o *Java*. Obviamente, porque no uso do C++ é necessário

incluir todas as classes no código transferido. O uso do *Java* pode ser muito vantajoso de for considerado o carregamento de aplicações em tempo de execução. Isto abre a interessante possibilidade do uso de agentes móveis [11] nestes sistemas. O uso de agentes móveis ou execução remota de aplicações implica conectividade com a rede. Assim, um segundo exemplo, que consiste numa típica aplicações de rede foi desenvolvida. Dado um endereço IP enviado pela porta série, o JUMPtec devolve o *host name* correspondente. Este pequeno exemplo foi desenvolvido apenas para demostrar o funcionamento do *Java* com a placa de rede.

Conclusões

Este trabalho mostra que é possível integrar a máquina virtual e restantes componentes *Java* num sistema de recursos limitados, como os usados para controlo e instrumentação. Neste caso especifico a máquina virtual foi integrada e testado num sistema com um processador 386Sx, com 2 Mb de memória e 2 Mb de disco. O sistema *Java* ocupa apenas 850Kb, deixando espaço suficiente para a existência de outros programas.

Neste artigo .dois exemplos simples foram apresentados e discutidos. Algumas medidas práticas foram efectuadas para avaliar o desempenho do sistema, concluindo-se que o *Java* leva a programas mais lentos, no entanto o tempo de resposta obtido é aceitável para muitas aplicações típicas do sistema.

Deste trabalho também se pode concluir que o *Java* oferece muitas funcionalidades, tais como rápido desenvolvimento, tamanho pequeno da aplicação e resposta temporal suficiente.

O trabalho futuro deve ser centrado no desenvolvimento de aplicações usando a rede, e estudar a possibilidade do uso de agentes móveis nestes sistemas.

Referências

[1] *WEBtoNET technical Manual*, JUMPtec Industry, 1997.

[2] Eric Giguère, *Java 2 Micro Edition*, John Wiley & Sons, Inc, 2000

[3] Java Web page, http://java.sun.com

[4] Dr- DOS web page, http://drdos.com

[5] "KVM porting guide", Sun Microsystems, Inc, 2000.

[6] Delorie Web Page, http://www.delorie.com

[7] Tim Lindholm, Frank Yellin, *The Java Virtual Machine Specification* , Addison-Wesley, 1996.

[8] "The K Virtual Machine (KVM), a white paper", Sun Microsystems, 1999.

[9] wattcp web page, www.wattcp.com

[10] Qusay H. Mahmoud, "MobiAgent: A mobile Agent Based Approach to Wireless information systems", School of Computer Science, Carleton University, Canada.

[11] V. Pham and A. Karmouch, "Mobile Software Agents: An Overview", *IEEE Communications*, Vol. 36, No. 7 (1998) pp. 26-37.

[12] V. Silva, J.L. Oliveira, J.A. Fonseca, "Ambiente de execução de aplicações Java para sistemas de recursos limitados", *CRC'2001*, 29-30 Novembro 2001, Covilhã.