O Movimento pelo *Software* Livre e a Plataforma Java

FERNANDO SILVA LOZANO[1]
HEITOR AUGUSTUS XAVIER COSTA[1]

[1]Curso ARL - DCC / UFLA - Cx Postal 3037 - CEP 37200-000 Lavras (MG)
fernando@lozano.eti.br,heitor@ufla.br

Resumo: *Este artigo argumenta sobre os motivos pelos quais é interessante e mesmo necessária a existência de uma implementação completamente* software *livre da plataforma Java. Mostra as dificuldades enfrentadas pelos projetos que buscam construir esta implementação, fazendo uma avaliação do estado-da-arte dos projetos de* software *livre voltados para a implementação do JavaSE.*

Palavras-Chave: *Java, software livre, licenças de software, Classpath.*

1 Introdução

A plataforma Java é tida como sendo a plataforma preferencial para o desenvolvimento de Sistemas de Informação nas empresas e também para a construção de *software* embarcado. Graças ao esforço do seu criador, a Sun Microsystems, e outras empresas e indivíduos associados ao JCP (Java Community Process), a plataforma Java goza de *status* de "padrão da indústria". Dentro do universo do *software* livre, a importância da plataforma Java pode ser comprovada por várias evidências, como: o fato de ser a linguagem de programação mais popular no SourceForge[1] ou pelo fato da própria Free Software Foudation considerá-la como componentes prioritários, para o Projeto GNU, o GCJ e o GNU Classpath[2].

Este artigo busca esclarecer o complicado relacionamento entre a plataforma Java e o *software* livre (*free software*) ou *software* de código aberto (*open source software*). Em especial, será argumentado que:

1. A plataforma Java e o *software* livre são não apenas compatíveis entre si, mas também complementares. A união dos dois traz mais benefícios para usuários e desenvolvedores de computadores do que a soma dos benefícios de cada um isoladamente.
2. Não existem impedimentos técnicos ou legais para a existência de uma implementação totalmente *software* livre da plataforma Java, que poderia inclusive ser oficialmente certificada como compatível com o padrão Java conforme a definição da Sun e do JCP.

[1] http://www.sourceforge.net

[2] Referências específicas às empresas, organizações, produtos e projetos citados nesta introdução serão apresentadas no decorrer deste artigo.

3. Os projetos de *software* livre que visam fornecer esta implementação Livre da Plataforma Java estão em estágio avançado, sendo viável sua utilização em produção para vários perfis de aplicação. Embora nenhum deles estejam ainda em condições de passar nos testes de certificação do JCP, vários deles estão bem perto de chegar a este ponto.

A metodologia adotada neste trabalho foi a pesquisa bibliográfica, análise da documentação fornecida pelos projetos de *software* livre em questão, além de documentos do JCP e diversos *sites* dedicados as comunidades de usuários/desenvolvedores Java. Esta pesquisa foi complementada pelo acompanhamento das listas de discussão dos projetos e comunidades, para avaliar o nível de qualidade e atividade nestes projetos, e a real demanda da comunidade de usuários Java por implementações livres da plataforma. O autor também instalou e experimentou o código fornecido por cada projeto, fazendo uma avaliação subjetiva da viabilidade de usar estes projetos, baseado em experiência de mais de 10 anos como consultor em projetos de desenvolvimento de aplicações Java e suporte à infra-estrutura de produção para estes sistemas.

A Seção 2 deste artigo esclarece o relacionamento entre a plataforma Java, seus padrões e o *software* livre (considerado, para efeitos práticos, como sinônimo de *software* de código aberto). A Seção 3, argumenta a favor da importância dos projetos de Java Livre, enquanto a Seção 4 avalia a viabilidade destes projetos atingirem suas metas. Na Seção 5 são apresentados e avaliados os principais projetos de Java Livre em desenvolvimento ativo como alternativas às implementações proprietárias oferecidas pela Sun e seus licenciados. Por fim, a Seção 6 reforça a viabilidade e necessidade de implementações Livres da plataforma Java.

2 Java é ou não é Software Livre?

É discutido dentro da comunidade de usuários e desenvolvedores de *software* livre, bem como entre os desenvolvedores Java em geral, se a linguagem Java é ou não é *software* livre. Essa discussão, tem sido caracterizada pela predominância de argumentos inconsistentes com os fatos e alto conteúdo emotivo dos participantes de ambos grupos. Entretanto, tanto o grupo que defende que a linguagem Java é sim *software* livre e deveria ser adotada em larga escala dentro da comunidade, quanto o que argumenta que Java não é Livre e sim *software* proprietário, por isso, deveria ser evitada e sofrer oposição pela comunidade, tem um certo grau de razão em seus argumentos. Ambos estão corretos em alguns pontos importantes e chegam a conclusão opostas por limitarem-se a analisar apenas alguns aspectos da questão.

A coerência foi prejudicada por empresas que entraram nele para atender a objetivos de *marketing*, por exemplo quando a IBM publicou uma carta aberta pedindo à Sun que transformasse o Java em *software* de código aberto[3]. O resultado foi o aumento do nível de desinformação circulando na comunidade sobre o tema. Apesar de afirmações equivocadas, feitas por alguns dos principais líderes da comunidade, como: Eric Raymond[4],

[3]Dois endereços onde encontram disponíveis esta carta: `http://lxer.com/module/newswire/view/5804/` e `http://www.it-director.com/article.php?articleid=11723`.

[4]"*Java is not only still a minority language in the open-source community but that it has not gained significant mindshare there since 1997[1]*".

a linguagem Java é uma parte importante e crescente da ecologia do *software* livre [2]. Como pode ser verificado em 2005 o Java tornou-se a linguagem mais popular do SourceForge[5], reconhecido como a maior incubadora de software Livre e aberto do mundo. Além disso, a maioria dos projetos da Apache Software Foundation (ASF)[6] são escritos em Java, especialmente os projetos: DB, XML e Jakarta.O que seria suficiente para justificar ações mais focadas da comunidade, em busca de uma implementação Livre desta linguagem. A própria FSF vem conclamando a comunidade para esta ação[7].

No final das contas, os fatos indiscutíveis são que Java, tomada como linguagem de programação e padrão de plataforma de desenvolvimento, não pode ser *software* livre, da mesma forma que a linguagem C em si, ou o protocolo TCP/IP, não podem ser por si só definidos como sendo *software* livre e nem como não sendo. A razão é bem simples: linguagens de programação, especificações de bibliotecas de programação e de protocolos, não são *software* de computador. São na verdade as especificações que permitem a construção dos compiladores, interpretadores e outros programas que visam fornecer os recursos específicos previstos por estas especificações [3].

Implementações específicas do Java podem ser ou não *software* livre. A implementação mais conhecida e utilizada, formada pelo JRE (Java Runtime Environment) e JDK (Java Development Kit) disponíveis para *download* gratuito pela Sun Microsystems[8], não é *software* livre. Muitos confundem esta implementação com a linguagem e a plataforma Java. Ela é licenciada para uso irrestrito em relação à quantidade de usuários ou de computadores, ou ao caráter comercial ou acadêmico dos usuários, e isenta da cobrança de *royalties* para a sua redistribuição junto com outros programas de computador. Apesar dessas liberdades, nenhuma das licenças da Sun para o Java (SCCL e JRL)[9] adequa-se à definição de *software* livre publicada pela FSF [4]. As licenças do Java também não se adequam a nenhuma outra definição alternativa de *software* livre ou *software* de código aberto, por exemplo o Contrato Social Debian [5] ou a Definição de *software* de código aberto da OSI [6].

Elas falham em ser licenças de *software* livre porque restringem o uso dos fontes, e não permitem *forks*. Apenas a própria Sun pode distribuir versões modificadas. Desenvolvedores estão sujeitos a NDAs (*Non-Disclosue Agreements*) se desejarem acesso aos fontes. E redistribuidores não podem redistribuir também *softwares* concorrentes, como implementações Livres do Java. Na verdade, as licenças da Sun são tão restritivas que nenhuma distribuição do Linux, mesmo as que permitem a inclusão de *software* proprietário, distribuem a implementação da Sun ou dos seus licenciados.

A maioria das implementações alternativas em uso no mercado, como as fornecidas pela IBM[10], BEA (JRockit)[11] e Blackdown[12], também não são *software* livre, uma vez

[5] `http://www.eweek.com/article2/0,1759,1896196,00.asp`.

[6] `Http://www.apache.org`.

[7] A *home-page* do Projeto GNU (`http://www.gnu.org`) em Março de 2006 contém, traz em destaque: "*Take action: Long-term contribution: Contribute to these high priority projects: 3D driver for ATI graphics cards, Gnash (the GNU Flash Player), GNU Compiler for Java, and GNU Classpath.*" [destaque pelo autor]

[8] `http://java.sun.com`.

[9] `http://java.sun.com/j2se/1.5.0/source_license.html`.

[10] `http://www-128.ibm.com/developerworks/java/jdk/`.

[11] `http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/jrockit/`.

[12] `http://www.blackdown.org`.

que estão sujeitas aos mesmos termos de licença da implementação da Sun. Na verdade, a totalidade das implementações certificadas pelo JCP[13] são derivadas da base de código original da Sun e por isso carregam essencialmente os mesmos termos de licença. A compatibilidade da Plataforma Java em si com o *software* livre é comprovada pela existência de vários projetos dentro da comunidade de *software* livre, cujo objetivo é fornecer uma implementação completa da plataforma Java, mais especificamente do JavaSE (*Standard Edition*).

Como exemplos de projetos de *software* livre que fornecem implementações da plataforma Java podem ser citados: Kaffe[14] e Harmony[15], este último patrocinado pela ASF, que é um dos membros mais importantes do JCP. Há também projetos focados em partes constituintes do JavaSE, como a máquina virtual Java (JamVM[16], SableVM[17]), a biblioteca de classes (Classpath[18]), ou o compilador Java (Jikes[19]). Outros projetos, ainda, fornecem implementações completas e certificados da plataforma JavaEE (Enterprise Edition), que seriam completamente livres se não exigissem como pré-requisito uma das implementações certificadas (hoje proprietárias) do JavaSE para sua execução, por exemplo JBoss[20], JonAS[21] e Gerônimo[22].

3 Sobre a Necessidade de um Java Livre

Grupos dentro da comunidade de *software* livre têm uma certa aversão ao Java. Às vezes alegando motivos técnicos, ("o Java é pesado e lento") senão, em uma manifestação "anti-corporativa", pela associação do Java a nomes como: Sun, Oracle, Borland e IBM. É uma reação psicológica semelhante à postura anti-Microsoft muito comum na comunidade. Mas, a tecnologia Java é bem diferente de outras tecnologias oriundas do mundo do *software* proprietário. Ela é desenvolvida sob os auspícios de um organismo chamado JCP, que reúne tanto empresas fornecedoras, quanto usuárias de tecnologia Java, além de membros individuais. Qualquer um pode associar-se ao JCP e participar do processo de definição de padrões para a plataforma Java.

O JCP foi muito criticado por falta de transparência ou por inviabilizar premeditadamente a existência de implementações abertas dos seus padrões. Por exemplo, a maioria dos TCKs[23] do JCP eram fornecidos sob licenças restritivas, vinculadas a NDAs e pagamento de *royalties*. Entretanto, desde o ano de 2002 o organismo vem modificando suas normas internas de funcionamento e tornando-se mais amigável à comunidade de *software* livre[24]. Pode-se creditar grande parte desta mudança de postura ao trabalho da

[13] `http://www.jcp.org`.
[14] `http://www.kaffe.org`.
[15] `http://incubator.apache.org/projects/harmony.html`.
[16] `http://jamvm.sf.net`.
[17] `http://sablevm.org`.
[18] `http://www.classpath.org`.
[19] `http://jikes.sourceforge.net/`.
[20] `http://www.jboss.org`.
[21] `http://forge.objectweb.org/projects/jonas`.
[22] `http://geronimo.apache.org/`.
[23] *Test Compatibility Kit*, testes automatizados para validar a aderência ou compatibilidade de um *software* com uma JSR.
[24] `http://www.theregister.co.uk/2002/11/28/ground_work_laid_for_open/`.

ASF, que faz parte do conselho diretor do JCP e lidera vários dos JSR (Java Specification Requests), que são os grupos de trabalho os responsáveis pela definição dos padrões e testes de compatibilidade oficiais do JCP.

Mas é necessário reconhecer que o trabalho do JCP permitiu a concorrência entre vários fornecedores de tecnologia Java, fornecendo produtos compatíveis e de aplicação, ferramentas de desenvolvimento, componentes, aplicações e *middleware*, que são segmentos de mercado bem mais visíveis e lucrativos, são outros membros do JCP. Em geral, A IBM, Oracle e BEA são reconhecidas como as líderes do mercado Java, com participação significativa também da: Borland, Macromedia, SAP, Nokia entre outras.

É indiscutível a popularidade do Java como plataforma para o desenvolvimento de Sistemas de Informação corporativos. Também há evidência em relação a aceitação do Java entre os desenvolvedores de *software* livre aberto. Mas, justamente essa aceitação do Java é um problema, pois ela cria uma "armadilha" que aprisiona grande quantidade de usuários e desenvolvedores em uma plataforma proprietária. O problema surge da necessidade, real ou percebida, de usar uma implementação Java proprietária para executar as aplicações livres escritas em Java [7].

Então, a força do Java, no mercado corporativo, é uma ameaça para os ideais do movimento pelo *software* livre. O Java tem o potencial de anular a maior parte dos ganhos obtidos pelo: GNU, Linux, BSD, PHP, Firefox e outros projetos construídos durante décadas pela comunidade. Em vez da dependência em relação à Microsoft, para a qual hoje o usuário doméstico já tem alternativas viáveis, há o risco de uma dependência em relação ao Java da Sun, especialmente para o usuário corporativo.

Para o usuário corporativo, sempre existe o risco da Sun modificar os termos de licença do Java, impor *upgrades* incompatíveis retroativamente e minar fornecedores concorrentes. Estrategicamente falando, seria bem melhor ter uma implementação do Java que não estivesse nas mãos de uma única empresa. Não é por coincidência que a "campanha de *marketing*" da IBM pela abertura do Java ocorreu em um momento durante o qual as ações da Sun Microsystems estavam desvalorizando-se e o mercado especulava sobre a viabilidade da empresa a longo prazo.

A existência de implementações livres do Java é também necessária para que a plataforma Java seja realmente universal. Apesar da promessa da Sun de construir uma plataforma WORA[25], na verdade é mais difícil encontrar implementações do Java maduras e mantidas ativamente para várias plataformas do que implementações de outras linguagens, como: Perl e PHP.

No caso dos sistemas operacionais livres existem implementações ou portes do Java da Sun apenas para processadores Intel (Sun, IBM e BEA) e PowerPC (IBM). Usuários de Linux em outras arquiteturas, como: SPARC e ARM, não tem acesso a uma máquina virtual Java certificada para rodar aplicações. Já usuários do FreeBSD estão ainda presos na versão 1.3.1[26]; enquanto a versão atual do Java é a 1.5.0 e espera-se para o meio de 2006, o lançamento da versão 1.6.0. Mesmo em sistemas proprietários, por exemplo no MacOS ou OpenVMS, as implementações certificadas disponíveis estão várias versões

[25] *Write-One, Run Anywhere* – escreva uma única vez e depois rode em qualquer lugar.

[26] `http://www.freebsd.org/java/` (versões mais recentes do que a 1.3.1, ainda estão em estágios alfa ou beta).

atrasadas, ou então demoram vários meses para serem colocadas em pé de igualdade com as versões do Java para Windows e Linux em plataforma Intel.

4 A Viabilidade do Java Livre

As recentes modificações na forma de trabalho do JCP tornaram legalmente possível existência de implementações livres dos seus padrões, certificadas e oficialmente reconhecidas como implementações compatíveis[9] e com direito de uso da marca Java. Embora, a certificação não signifique muito para o desenvolvedor de *software* livre, ela é essencial para a aceitação de implementações livres da plataforma Java no meio corporativo, como demonstra o caso do JBoss[27]. Certificar uma implementação livre do Java, entretanto, não é um processo simples. O problema do custo foi de certa forma resolvido quando o JCP estabeleceu um programa de "bolas" para financiar a certificação de projetos de *software* livre[28], e pelo patrocínio de empresas do porte da Intel e IBM a projetos como o Harmony[29].

Discussões nas listas do Harmony e do Kaffe indicam as duas maiores dificuldades. Primeiro, a licença de uso do TCK para o JavaSE ainda possui cláusulas de não-divulgação que deixam dúvidas se alguém que tenha acesso ao TCK poderá participar do desenvolvimento de um projeto de Java Livre, sem o risco de "contaminar" o projeto, deixando seus desenvolvedores sujeitos a ações judiciais por violação de contrato[30]. Estreitamente relacionado com este problema são os termos da SCCL e JSRL em si. É consenso dentre os projetos de Java Livre que eles não podem aceitar contribuições de desenvolvedores que foram expostos ao código da Sun, pois os termos destas licenças também incluem várias cláusulas de sigilo e de exclusividade[31].

Outro problema decorre do fato da maioria dos projetos de Java Livre ter nascido de forma independente, não-coordenada e com um escopo de atuação limitado. Isto fez com que cada um adotasse uma licença diferente e muitas vezes não é possível combinar componentes que juntos poderiam formar uma implementação livre do JavaSE. A incompatibilidade, em geral, envolve projetos desenvolvidos sob as licenças GPL e LGPL, contra projetos desenvolvidos usando licenças estilo BSD, como: a APL da ASF e a EPL da Eclipse Foundation.

No final das contas, as licenças do JCP e as próprias licenças de *software* livre impedem que o grande contingente de desenvolvedores que possuem conhecimentos sobre como implementar diferentes partes da plataforma Java seja facilmente reunido para a criação de uma implementação livre e robusta do Java. Qualquer projeto deste tipo tem que garantir antes do *status* de *clean-room*. Os desenvolvedores do GNU, do Linux, do GCC

[27] O JBoss é um servidor de aplicações JavaEE Livre, que vinha há alguns anos tornando-se a implementação preferencial para desenvolvimento ou produção em pequena escala. Apesar das afirmações dos seus desenvolvedores de que a certificação pelo JCP era irrelevante para seus usuários, ele só obteve aceitação entre grandes empresas depois que a empresa que reúne seus desenvolvedores, o JBoss Group, entrou no JCP e certificou seu produto livre.

[28] `http://news.com.com/2100-1001-949591.html`.

[29] `http://news.com.com/IBM+steps+into+open-source+Java+project/2100-7344_3-5798290.html`.

[30] `http://www.advogato.org/person/robilad/diary.html?start=64`.

[31] `http://blog.netbeans.jp/roller/page/fchoong/20040414#james_gosling_father_of_java`.

não tiveram este problema, pois desenvolvedores que antes atuaram em projetos proprietários relacionados com Unix e Linguagem C, não estavam sujeitos a NDAs tão rígidos quanto os do Java. Outros projetos de *software* livre tem total liberdade em aceitar contribuições vindas de pessoas que um dia já foram (ou ainda são) funcionárias de empresas desenvolvedoras de *software* proprietários similares.

5 Avaliação do Estado-da-arte do Java Livre

Mesmo com as dificuldades, há uma grande quantidade de projetos voltados para implementações livres da plataforma Java. Vários destes projetos estão bem próximos de oferecer uma alternativa real à implementação da Sun e suas derivadas[32].

5.1 Kaffe

O progresso do Java Livre acelerou de forma significativa quando Dalibor Topic assumiu a manutenção do Kaffe em 2002 e buscou ativamente a colaboração com outros projetos de *software* livre relacionados com o Java [10], que antes atuavam como ilhas isoladas de conhecimento. O Kaffe é o projeto mais antigo de implementação do Java fora da Sun, tendo sido criado por Tim Wilkinson em 1996. Esse autor no mesmo ano fundou a empresa Transvirtual, que intencionava vender serviços de suporte ao uso da tecnologia para aplicações embarcadas, de modo similar ao feito com sucesso pela Cygnus Solutions, (hoje parte da Red Hat) com o GCC. Apesar do sucesso do Kaffe com a comunidade acadêmica, a Transvirtual não teve muito sucesso no mercado, devido a dificuldade em acompanhar o ritmo de evolução da implementação da Sun, e os novos padrões definidos pelo JCP. Entre 2000 e 2001, o projeto estava inativo.

Quando Dalibor assumiu o projeto, ele primeiro buscou ajustar o processo de compilação do Kaffe e incorporar os vários *patches* que estavam pendentes de avaliação. Dalibor percebeu que seria inviável continuar evoluindo com um projeto tão grande dentro de uma comunidade tão pequena. Então, ele decidiu concentrar-se na JVM em si, buscando o apoio do projeto GNU Classpath para manter as bibliotecas de classes. Mesmo dentro da própria JVM, ele buscou idéias e código de outros projetos, como o GCJ ou a pesquisa sobre coletores de lixo de Hans Boehm da HP. Também facilitou a integração do Kaffe, que seria apenas uma alternativa ao JRE, com compiladores externos, como o Jikes, de modo a formar também uma alternativa ao JDK.

Como resultado, não apenas o Kaffe foi revigorado, mas toda a comunidade de projetos de *software* livre para substituir o Java da Sun ganharam impulso. Em 2005, o projeto Kaffe candidatou-se a uma bolsa do JCP, para o processo de certificação. Com esse processo passou a receber o apoio do SouJava, um grupo de usuários Java brasileiro que é considerado o maior do mundo.

5.2 GCJ

O GCJ (GNU Compiler for Java) é parte do conjunto de *software* conhecido como GCC (GNU Compiler Collection). O GCC é reconhecido como um dos melhores compiladores

[32] Veja, por exemplo, a comparação em termos de classes e métodos fornecidos pelo Classpath e pelo Java da Sun em `http://www.kaffe.org/stuart/japi/htmlout/h-jdk14-classpath.html`.

C do mercado e certamente com suporte mais amplo a diferentes arquiteturas de processadores e sistemas operacionais [11]. A arquitetura do GCC prevê um *back-end* comum de geração e otimização de código, que é compartilhado por vários *front-ends* resposáveis pela análise léxica e sintática de diferentes linguagens de programação C. O C++, ObjectiveC, Ada, Fortran, e Java são as linguagens suportadas pela distribuição oficial do GCC, mas existe uma grande quantidade *front-ends* para outras linguagens desenvolvidas por terceiros.

Note que o foco do GCJ é radicalmente diferente da plataforma Java conforme implementada pela Sun. Em vez de compilar para *bytecodes* que são depois executados por uma JVM. o GCJ compila diretamente para o código nativo do processador escolhido, o que apresenta vantagens significativas para o uso embarcado. Entretanto, o GCJ também pode ser usado como um compilador Java tradicional, pois para o engine de geração de código do GCC, os *bytecodes* são como a linguagem de máquina de um processador qualquer.

Completando o conjunto, o GCJ inclui um interpretador de *bytecodes* chamado GIJ (GNU Interpreter for Java), que efetivamente é uma JVM. O GIJ pode ser embutido na biblioteca de tempo de execução do GCJ, a libgcj, e assim é possível misturar em uma mesma aplicação classes compiladas para *bytecodes* e classes compiladas nativamente. Em grandes aplicações *desktop* ter as classes que forma a API do JavaSE compiladas nativamente reduz significativamente o tempo de início de uma aplicação, e também o seu consumo de memória. A dupla GCJ/GIJ não inclui um JIT (*Just-In-Time compiler*), componente que transforma os *bytecodes* dos métodos mais usados em código nativo, para aumentar a velocidade de execução. Isto significa que o GIJ será sensivelmente mais lento do que o Java da Sun ou mesmo outras implementações livres do Java que possuam um JIT, como: o Kaffe.

Assim como o Kaffe, o GCJ optou há cerca de dois anos por interromper o desenvolvimento da sua própria biblioteca de classes para o JavaSE e importar estas classes do projeto GNU Classpath. Na verdade pode-se dizer que foi a aglutinação dos projetos em torno do Classpath que permitiu a todos saírem do estado de lertargia no final do ano 2002 e tornarem-se desde 2005 em alternativas reais ao Java proprietário.

Hoje o GCJ é a única implementação livre do Java para usuários de sistemas proprietários como o Windows. Ele faz parte do MingW[33], porte nativo[34] do GCC para Windows. O GCJ é a implementação livre do Java de maior sucesso até o momento, pois é parte integrante das versões mais recentes da distribuição Fedora Core e Debian. Ele tem sido utilizado com sucesso para incluir nestas distribuições programas livres populares entre os desenvolvedores Java, como: o Tomcat, Eclipse, Ant e JUnit. Estas distribuições fornecem, com base no GCJ, um ambiente completo e autônomo para o desenvolvimento e execução de aplicações Java, especialmente se o foco for o desenvolvimento *web*.

Recentemente, os desenvolvedores do GCJ contemplam eliminar o seu *front-end* de análise léxica da linguagem Java e incorporar o ECJ, parte do IDE Eclipse. Isto porque estaria mais evoluído em relação às novidades introduzidas na linguagem pelo JDK 1.5.0 (Tiger) da Sun, e as novidades esperadas para o JDK 1.6.0 (Dolphin). Não existe uma

[33] `http://mingw.sf.net`.

[34] Não confundir com o Cygwin, que usa um ambiente de emulação de Unix para rodar o GCC em Windows.

posição oficial do comitê gerenciador do Projeto GCC sobre a viabilidade legal ou o interesse estratégico para o Projeto GNU em se efetivar esta união[35].

5.3 JamVM

Outra alternativa é a JamVM que diverge da arquitetura da JVM proprietária da Sun. Em vez de usar o processo de JIT para obter performance, a JamVM opta por melhorar a qualidade do seu interpretador. JVMs proprietárias escritas para uso embarcado seguem a mesma estratégia. Nem a KVM da Sun ou a J9 da IBM incluem um JIT, pois o consumo de memória e processador seriam proibitivos em celulares e PDAs, que são justamente o foco do JamVM.

O JamVM suporta arquiteturas não-Intel como PowerPC e ARM, que são populares em aplicações embarcadas, e faz parte também da distribuição Familiar Linux[36], voltada para uso em PDAs das linhas: iPAQ e Zaurus. O JamVM também tem tornado popular por seu suporte "*plug-and-play*" ao GNU Classpath. Em vez de importar as classes deste projeto e fazer adaptações para adequá-las ou otimizá-las para as características específicas da JVM, como é feito pelo Kaffe e GCJ, o JamVM usa as classes inalteradas. Então, quando é liberada uma nova versão do Classpath, ela pode ser imediatamente testada com o JamVM, em vez de aguardar pela atualização de outro projeto de JVM Livre.

5.4 Apache Harmony

O Harmony [13] é o projeto mais novo dentre as máquinas virtuais Java Livres. Apesar disso, é o que tem recebido maior apoio corporativo, com o envolvimento direto da IBM e Intel no projeto. A razão disso é o fato da maioria dos outros projetos utilizarem a licença GPL, que não é considerada *business-friendly*. Embora o uso da GPL não impeça o uso de JVMs Livres para executar aplicações Java proprietárias, impede o uso destas JVMs como ponto de partida para produtos proprietários, por exemplo JVMs a serem embutidas em celulares, ou em servidores de alto desempenho.

Além disso, a questão de incompatibilidade entre a GPL e outras licenças de *software* livre impede o Harmony de utilizar o GNU Classpath e outros componentes que são integrados ao Kaffe e GCJ. Então pode-se esperar que o Harmony demore vários anos para tornar-se usável, apesar do seu patrocínio mais forte e de ser o único de todos os projetos de Java Livre a receber manifestações de apoio da Sun.

Por outro lado, o Harmony tem "queimado etapas" no seu desenvolvimento graças à doações de código da IBM, HP e Intel[37]. Mas estas doações são esporádicas e fornecem sempre componentes incompletos, pela necessidade de se garantir a independência em relação ao código oriundo da Sun[38]. O grande problema são os NDAs, que impedem a atuação das pessoas que estiveram envolvidas antes com os projetos de Java proprietário. O Harmony pode dar um grande salto se for aceita e integrada a contribuição do grupo que desenvolve a SableVM[39]. Esta JVM era desenvolvida originalmente sob a GPL, e,

[35] `http://gcc.gnu.org/ml/java/2006-04/msg00004.html`.
[36] `http://www.handhelds.org`.
[37] `http://www.theinquirer.net/?article=30688`.
[38] IBM, Intel e HP já forneciam antes produtos baseados no Java da Sun
[39] `http://www.sablevm.org`.

como a maioria das outras JVMs Livres, baseada no Classpath para prover as APIs do JavaSE.

Em alguns aspectos, o SableVM encontra-se mais evoluída do que outras JVMs Livres, por ter uma implementação do JDWP (que permite a depuração de código Java pela definição de *breakpoints* e inspeção do estado dos objetos em memória). O grupo principal de desenvolvedores aprovou o re-licensiamento do código sob a APL e a submissão às normas do PMC do Projeto Harmony, de modo que o SableVM deixa de ser um projeto independente e funde-se ao Harmony[40].

5.5 Outras JVMs Livres

Outros projetos de JVMs Livres que mereceriam uma análise mais profunda:

- CacaoJVM[41] serve de base para pesquisas sobre novos algoritmos internos de funcionamento de uma JVM e vem sendo cada vez mais a plataforma preferida para o desenvolvimento do próprio GNU Classpath;
- IKVM[42] permite rodar *bytecodes* Java dentro do CLR do .NET da Microsoft ou do seu "*clone*" criado pelo projeto Mono, possibilitando a inteproperabilidade transparente entre classes Java e classes C# ou VB.NET e vice-versa;
- JikesRVM[43] é totalmente implementada em Java, e nem por isso torna-se mais pesada ou lenta do que as JVMs implementadas em C, demonstrando que o Java poderia suportar o seu próprio desenvolvimento como acontece com o Unix e a linguagem C;
- Mysaifu[44] suporta JavaSE completo, e não o limitado JavaME, em PDAs baseados no sistema operacional PocketPC / WindowsCE.

O CacaoVM, IKVM e Mysaifu também incorporam GNU Classpath como implementação das APIs do JavaSE.

5.6 GNU Classpath

O GNU Classpath [14] nasceu com o objetivo de fornecer implementações livres para várias bibliotecas de extensão que estavam sendo propostas nos primórdios da plataforma Java. Estava claro que várias aplicações seriam inviáveis como *software* livre se dependessem do código das implementações de referência fornecidas pelo JCP na época. A própria Apache Software Foundation foi usuária do Classpath nos seus primeiros servidores de aplicação Java, como o JServ[45], que depois foi descontinuado em favor do Tomcat. Logo, o escopo do projeto Classpath cresceu para a re-implementação da biblioteca básica de classes do JavaSE, um processo lento, bem mais complexo do que a implementação de uma JVM e lutando com a constante atualização do Java da Sun.

[40] `http://sablevm.org/lists/sablevm-devel/2006-March/000608.html`.
[41] `http://www.cacaojvm.org/`.
[42] `http://sourceforge.net/projects/ikvm`.
[43] `http://jikesrvm.sourceforge.net/`.
[44] `http://www2s.biglobe.ne.jp/dat/java/project/jvm/index_en.html`.
[45] `http://archive.apache.org/dist/java/`.

Apenas depois que os principais projetos de JVMs Livres decidiram concentrar seus esforços no Classpath, que o progresso começou a ser visível. No início de 2006 mais de 95%[46] de todo o JavaSE 1.4 estava implementado no Classpath, além de grande parte do 1.5 (que estava em uma linha de desenvolvimento separado devido à necessidade de mudanças nas JVMs e compiladores para suportar esta versão). No final de Março de 2006 se havia atingido 98% do JavaSE 1.4 implementado. Hoje, as principais deficiências no Classpath estão no suporte a aplicações gráficas Swing e comunicação via CORBA. Então, há uma grande quantidade de aplicações Java populares que podem ser executadas apenas com as classes fornecidas pelo Classpath, inclusive aplicações gráficas baseadas no SWT, como o próprio Eclipse.

O percentual de implementação do Classpath em relação ao JavaSE é um tanto enganador: várias classes e métodos são apenas *stubs*, ou tem comportamento diferente do Java da Sun, e por isso, comprometem a execução de aplicações que dependem deles. Mais do que a imaturidade do Classpath, isto reflete a falha inerente ao conceito de que é possível haver um padrão real sem implementações independentes. Na maioria dos casos, as diferenças não são descritas claramente nos padrões do JCP, e em alguns casos o Java da Sun viola as especificações formais definidas por JSRs que ela mesmo havia liderado!

5.7 Jikes e Eclipse

Além de JVMs e APIs do JavaSE, um ambiente de desenvolvimento Java necessita de um compilador. Hoje, os dois mais aceitos são projetos nascidos da IBM. O Jikes era uma plataforma de pesquisa para vários aspectos da linguagem Java, um compilador rápido e robusto que muitas empresas preferem usar em lugar do compilador fornecido pela Sun no seu JDK (Java Development Kit). Já o Eclipse[47], projeto para criação de um IDE universal, independente de linguagem de programação ou de sistema operacional, inclui um compilador Java incremental atualizado para as sintaxes mais novas do Java 1.5.

A Red Hat separou o compilador do Eclipse do restante do pacote, chamando o resultado de ECJ (Eclipse Compiler for Java). Essa separação objetiva contornar as dificuldades do GCJ em compilar grandes volumes de código como o próprio Eclipse e assim viabilizar a inclusão destes pacotes em sua distribuição, sem depender de *software* proprietário. O resultado desta separação vem sendo utilizado em outros projetos de *Software* Livre, por exemplo o container *web* Tomcat 5.5[48] incorpora o ECJ para a compilação de páginas JSP sem depender do JDK da Sun.

5.8 Testes com Mauve e Gump

As dificuldades da comunidade de *software* em conseguir o acesso aos TCKs do JCP levou a mesma, com apoio substancial da Red Hat e HP, a criar sua própria *suite* de testes. O Mauve[49] pode ser usado para testar tanto JVMs em si quanto implementações das APIs do JavaSE ou compiladores Java. Seu objetivo não é tanto testar a compatibilidade com os padrões do JCP (embora este seja um dos objetivos do projeto), mas principalmente

[46] `http://savannah.gnu.org/forum/forum.php?forum_id=4097`.
[47] `http://www.eclipse.org`.
[48] `http://tomcat.apache.org`.
[49] `http://sources.redhat.com/mauve/`.

garantir a qualidade e funcionamento dos componentes testados. Esforços da comunidade como o Mauve demonstram que os argumentos contra a abertura do Java baseados na possibilidade de haverem vários *forks* são infundados.

Na verdade, como explica Eric Raymond [13], a fragmentação em várias versões incompatíveis vai contra o espírito e as práticas das comunidades de desenvolvedores de *software* livre. As várias distribuições do Linux são freqüentemente citadas como exemplos de como o *software* livre poderia prejudicar a compatibilidade, mas Raymond demonstra que elas na verdade não são incompatíveis entre si, enquanto que padrões baseados em *software* proprietário como o Unix sim mostraram-se bastante incompatíveis.

Já o Gump[50] traz uma abordagem diferente para o teste de aplicações Java. Ele é baseado em *software* populares da ASF, como: Ant e Tomcat. Então desenvolvedores de JVMs e bibliotecas de classes podem ter *feedback* rápido quando suas modificações afetarem estas aplicações populares. O mesmo vale para desenvolvedores de bibliotecas usadas como infra-estrutura, por exemplo os Jakarta Commons[51].

5.9 JPackage

O Projeto JPackage[52] visa atender a uma necessidade mais "mundana" da comunidade de *software* livre: a falta de pacotes para a instalação do Java e de aplicativos Java nas distribuiçõs mais populares do Linux. Quando estes pacotes são fornecidos (a Sun por exemplo, fornece um pacote RPM para a instalação do Java em Linux) eles não seguem as boas práticas do ambiente, não obedecem ao LSB (Linus Standards Base), não configuram o acesso aos seus comandos pelo *shell* e não trazem a informação correta de dependências. Em suma, o administrador de sistemas Linux não se beneficia dos pacotes RPMs "oficiais" do Java. O projeto fornece ainda empacotamentos alternativos para as implementações proprietárias do Java[53] e para diversas bibliotecas e aplicações proprietárias populares, na seção "*non-free*"[54] do projeto. E, na sessão "*free*" são fornecidos pacotes para centenas de aplicações e bibliotecas livres.

Outro recurso interessante do JPackage é o mecanismo alternatives do Debian para permitir ao administrador optar por uma dentre várias implementações para a mesma especificação do mundo Java. Então, o administrador pode optar ora pelo Java da Sun, ora pelo GCJ, ou então definir para cada aplicação qual a sua JVM preferencial. Outro exemplo seria permitir ao desenvolvedor optar pelo Jessie[55] como implementação do JCE[56], em vez da implementação de referência da Sun, ou por uma implementação otimizada para um co-processador criptográfico.

Os pacotes do JPackage tem sido utilizados com sucesso em várias distribuições: Red Hat, Fedora, SuSE, Mandriva e outras baseadas em RPM. Muitas vezes eles têm sido adotados como parte oficial das distribuições, por exemplo no Fedora Core. E tem servido

[50] `http://gump.apache.org`.

[51] `Http://jakarta.apacje.org/commons`.

[52] `http://www.jpackage.org`.

[53] Na verdade, são fornecidas apenas instruções de como fazer o re-empacotamento de acordo com os padrões do Linux, devido às restrições das licenças do Java da Sun.

[54] De "*não-Livre*", ou seja, *softwares* proprietários.

[55] `http://www.nongnu.org/jessie`.

[56] Conjunto de classes que implementam algoritmos de criptografia

de referência mesmo para distribuições que não usam o RPM, pois o Debian está usando o trabalho do JPackage, em uma parceria de mão dupla, para incluir aplicações livres em sua distribuição.

5.10 Java-Gnome

O projeto Java-Gnome[57] traz uma alternativa bastante interessante para o desenvolvedor Linux. Afinal o padrão oficial da plataforma Swing ainda é bastante criticado devido à sua baixa performance e sua falta de integração com o *desktop* do usuário. Neste sentido, o Java-Gnome é um concorrente do SWT utilizado pelo Eclipse, com a vantagem de que o SWT, embora forneça fidelidade ao visual do ambiente nativo, não fornece integração real com este *desktop*. Por exemplo, o SWT não fornece acesso ao painel (barra de ferramentas) da área de trabalho ou associações de tipos de arquivos.

As bibliotecas do Gnome foram concebidas desde o início para serem portáveis, independente da linguagem de programação e do sistema operacional. Elas tem sido utilizadas com sucesso em linguagens variadas como: C, Pascal, Ada, PHP, Ruby e Python. Aplicativos Gnome populares como o Gimp, Planner e Ethereal têm sidio portados até para Windows. E a ferramenta visual para desenho de intefaces Gnome, o Glade, também busca ser independente de linguagem, graças ao uso de um formato XML para salvar a descrição das janelas.

O Java-Gnome não faz nada mais do que permitir o acesso às bibliotecas e ferramentas do Gnome para os programadores Java, por meio do JNI. O estilo de programação é natural para o desenvolvedor Java e segue as convenções universais do Gnome, preservando o conhecimento do desenvolvedor em outras linguagens, e garantindo que aplicações Java integrem-se naturalmente ao *desktop* do usuário GNU/Linux. Um benefício adicional do Java-Gnome é que ele permite agora mesmo o uso de JVMs Livres para desenvolver e executar aplicações gráficas em Java. Não há necessidade de esperar pelo amadurecimento da implementação do Swing, inclusa no Classpath. Neste caso, o SWT do Eclipse também é uma alternativa, mas o SWT não oferece integração com o Gnome.

6 Conclusão

É possível uma implementação Livre do Java dentro das normas atuais do JCP, coerente com a proposição de universalidade e interoperabilidade da plataforma Java, justificada não apenas por questões ideológicas. A própria diversidade de abordagens tecnológicas e de áreas de aplicações dos projetos relacionados com JVMs Livres demonstram que há espaço e demanda pelas implementações livres.

Além disso, a colaboração voluntária entre os projetos de Java Livre, juntamente com os esforços surgidos dentro da própria comunidade para validar formalmente a compatibilidade das JVMs Livres demonstram a incoerência do principal argumento contra o Java Livre: o argumento de que ele iria fragmentar a plataforma Java e acabar com o seu maior benefício, a garantia de compatibilidade.

Embora não tenham sido fornecido detalhes sobre os testes executados em cada projeto de Java Livre, os projetos citados foram instalados e executados com sucesso, utili-

[57] `http://java-gnome.sf.net`.

zando apenas os *downloads* e documentação disponíveis nos respectivos *sites*. A atividade em suas *maling lists* indica que as comunidades de usuários estão em franco crescimento, o que é confirmado pelo interesse demonstrado pela mídia técnica, por exemplo O'Reilly e Java.Net. Os projetos de Java Livre já atingiram maturidade suficiente para sua inclusão como parte oficial de várias distribuições do Linux, e assim não há motivos para que o desenvolvedor de aplicações evitar implementações livres do Java.

Referências

[1] ERIC S. RAYMOND. Let Java Go, Round 2. 2004. Disponível em: <http://www.catb.org/esr/writings/let-java-go-2.html>.

[2] TIM O'REILLY. The Rise of Open Source Java, 2005. Disponível em: <http://radar.oreilly.com/archives/2005/06/the_rise_of_ope.html>

[3] BRUNO SOUZA. A Luta pelo Java Livre, Revista Java Magazine n. 19, nov. 2004

[4] Free Software Foundation. O Que é Software Livre, 1996. Disponível em: <http://www.gnu.org/philosophy/free-sw.pt.html>

[5] DEBIAN. Contrato Social Debian, Versão 1.1 ratificada em 26 de Abril de 2004. Disponível em: <http://www.debian.org/social_contract>

[6] OpenSource Initiative OSI. The Open Source Definiton, 2006. Disponível em: <http://www.opensource.org/docs/definition.php>

[7] RICHARD STALLMAN. Free But Shackled - The Java Trap, 2004. Disponível em: <http://www.gnu.org/philosophy/java-trap.html>

[8] JAVANET. Java Research License. Disponível em: <http://java.net/jrl.csp>

[9] JCP. Proccess Document, Version 2.6, 2004. Disponível em: <http://jcp.org/en/procedures/jcp2>

[10] DALIBOR TOPIC Kaffe Past, Present and Future, 2005. Disponível em: <http://kaffe.org/doc/slides/kaffe-past-present-future-FOSDEM05.pdf>

[11] PER BOTHNER. Compiling Java With GCJ, Linux Journal, 2003. Disponível em: <http://www.linuxjournal.com/article/4860>

[12] RAYMOND, E. S. The Cathedral & The Bazaar. Sebastopol: O'Reilly, 1999. Disponível em: <http://www.catb.org/esr/writings/>

[13] HARMONY. Harmony Architecture, ASF, 2005. Disponível em: <http://wiki.apache.org/harmony/HarmonyArchitecture>

[14] CLASSPATH. What is GNU Classpath, FSF, 2006. Disponível em <http://www.klomp.org/mark/classpath/lt2006/paper.html>