Um estudo sistemático de licenças de software livre

Vanessa Cristina Sabino

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Ciência da Computação

Orientador: Prof. Dr. Fabio Kon

Durante o desenvolvimento deste trabalho a autora recebeu auxílio financeiro da CNPq

São Paulo, Agosto de 2011

Um estudo sistemático de licenças de software livre

Esta dissertação contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa realizada por Vanessa Cristina Sabino em 12/08/2011. O original encontra-se disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Prof. Dr. Fabio Kon (orientador) - IME-USP
- Prof. Dr. Daniel Macêdo Batista - IME-USP
- Prof. Dr. Juliano Souza de Albuquerque Maranhão - FD-USP

Agradecimentos

Ao meu orientador, Prof. Dr. Fabio Kon, pela sua dedicação a este trabalho.

Ao Prof. Dr. Juliano Maranhão e ao Prof. Dr. Augusto Marcacini, por me auxiliarem a compreender questões essenciais relacionadas ao Direito que foram levantadas nesta dissertação.

Ao Carlos Duarte do Nascimento (Chester), pelo incentivo e ajuda valiosa na revisão deste trabalho.

Aos meus colegas do IME, que me acompanharam desde a graduação, por transformarem o processo de aprendizado em uma verdadeira experiência de vida. Em particular ao grupo do Centro de Competência em Software Livre, cujas discussões foram fundamentais para servir de base para muitas das ideias expostas aqui.

Aos meus pais, pelo apoio demonstrado durante essa fase.

"Meaning lies as much
in the mind of the reader
as in the Haiku"

Douglas R. Hofstadter

Resumo

Esta dissertação tem por objetivo apresentar as licenças de software livre mais importantes, sob a luz dos seus principais aspectos jurídicos e da inter-compatibilidade, de forma a auxiliar pessoas envolvidas no desenvolvimento de software a compreender as implicações destas licenças ao fazer uso delas em seus projetos.

A dissertação contextualiza as licenças, tanto no tocante à legislação brasileira, quanto no que diz respeito às restrições de licenciamento, de forma a viabilizar a análise de compatibilidade que se segue. Casos de projetos proeminentes de software livre cujo desenvolvimento foi afetado pelas implicações mencionadas ilustram a investigação, que é complementada por uma análise de ferramentas e metodologias existentes que auxiliam na gestão dos aspectos de licenciamento.

Palavras-chave: software livre, licença, direito autoral, GPL, open source.

Abstract

The purpose of this Master thesis is to present the most common free software licenses, regarding their main legal and inter-compatibility aspects, to help people involved in software development understand the implications of these licenses when using them in their projects.

It contextualizes the licenses, both in terms of the Brazilian legislation, and regarding licensing restrictions, to make the subsequent compatibility analysis possible. Cases of free and open source software in which development was affected by the mentioned implications illustrate the research, and it is complemented by an analysis of existing tools and methodologies that assist in the management of licensing issues.
Keywords: free and open source software, license, copyright, GPL.

Sumário

Capítulo 1

Introdução

Programas de software livre em geral são de fácil acesso. Porém, a simples obtenção de um programa não significa que se possa fazer o que quiser com ele. As leis de direito autoral impõem várias restrições, sendo dependente de autorização prévia e expressa do autor diversas formas de utilização da obra, tais como cópia e reprodução. As licenças de software livre são contratos através dos quais os detentores dos direitos sobre um programa de computador autorizam usos de seu trabalho (dentro do que for permitido em lei) que, de outra forma, seriam exclusivos dos detentores.

Além da execução do programa como usuário final, esses usos autorizados permitem que desenvolvedores possam adaptar o software para necessidades mais específicas, utilizá-lo como fundação para construção de programas mais complexos, entre diversas outras possibilidades. Elliot e Scacchi [ES08] também observam que a licença, em muitos projetos, não é apenas uma questão de regras para uso da propriedade intelectual, mas também uma afirmação sobre os princípios e valores das comunidades que os desenvolvem e uma forma de se afiliar a um movimento social maior. Nesta dissertação, é apresentado como a escolha da licença influencia a forma como o software poderá ser usado, desenvolvido e distribuído.

A presente dissertação é um estudo sobre tipos de licenças de software que podem ser agrupadas em regime chamado "livre". Suas principais contribuições são:

- reunir, em um único local, informações sobre as principais licenças de software livre;
- contextualizar as licenças em relação à legislação brasileira;
- apresentar, de forma sistemática, as vantagens e desvantagens de cada licença;
- analisar a compatibilidade entre licenças de software livre.

O objetivo desta dissertação é auxiliar pessoas que não são especialistas em licenciamento de software a entender melhor suas implicações, tornando o texto das principais licenças mais acessível e facilitando o processo de escolha de software livre realizado por gerentes, investidores, desenvolvedores, estudantes, educadores, cientistas e demais pessoas envolvidas com software. Por outro lado, também esperamos tornar mais claro,

para juristas, aspectos do desenvolvimento e uso de software que são relevantes para o entendimento das licenças.

Cada licença será discutida de forma sistemática, traduzindo as principais ideias contidas em seus termos para uma linguagem mais familiar ao desenvolvedor de software, explicando quando seu uso é recomendado e apresentando as principais vantagens e desvantagens.

Antes disso, de forma a contextualizar o estudo, será apresentado um breve histórico do movimento de software livre e suas principais características. Também serão estudados os aspectos jurídicos envolvidos no licenciamento de software, dando atenção especial às diferenças existentes entre a legislação brasileira e a americana, visando ao entendimento de como as licenças escritas nos Estados Unidos se adequam ao nosso ordenamento.

Outro ponto que será discutido é a compatibilidade entre licenças, muito importante no processo de integração de diversos componentes para criação de projetos mais complexos. Serão avaliados diversos casos de uso e suas possibilidades de licenciamento. Também serão discutidas algumas ferramentas e processos que podem ajudar na tarefa de análise de compatibilidade de licenças em projetos de software.

Para finalizar, veremos, na forma de estudos de caso, alguns projetos que possuem um histórico de características interessantes quanto ao licenciamento.

1.1 Trabalhos Relacionados

Conforme será visto no próximo capítulo, o movimento de Software Livre é bastante recente, tendo surgido há menos de três décadas. Apesar de algumas das primeiras licenças terem sua origem em universidades, a pesquisa acadêmica sobre esse assunto era quase inexistente até muito recentemente.

Porém, na última década, em particular nos últimos cinco anos, à medida que crescia a diversidade de projetos e a adoção de software livre por parte das empresas, o tema ganhou maior importância e passou a ser estudado por diversos grupos, tanto na área da computação como também no direito. Em 2009 surgiu a primeira revista científica com foco nesse tema, a *International Free and Open Source Software Law Review* [Int09].

Alguns trabalhos nessa área merecem destaque. O primeiro é o artigo *Software Licenses in Context: The Challenge of Heterogeneously-Licensed Systems* (2010) [ASA10]. Sua proposta foi de construir um modelo teórico que permita analisar sistemas em que existe mais de uma licença envolvida e entender quais são as obrigações, direitos e conflitos. Segundo os autores, "nenhuma licença, tal como é, se aplica ao sistema resultante. Ao invés disso, existe uma coleção de direitos disponíveis para o sistema, e, para cada direito, existe uma obrigação correspondente que precisa ser observada. É possível que o conjunto de direitos seja vazio, e se direitos específicos são desejáveis, tais como o direito de usar o sistema e de distribuí-lo, esses direitos precisam ser considerados em cada estágio do desenvolvimento, desde o projeto até a distribuição. Os direitos específicos e obrigações são determinados pelas licenças dos componentes do sistema e pela configuração arquitetural

em que eles foram combinados".

Outro trabalho relacionado é o artigo *License Integration Patterns: Dealing with Licenses Mismatches in Component-Based Development* (2009) [GH09], em que, assim como no artigo acima, German e Hassan tratam do problema de combinar componentes de software que estão sob licenças diferentes. Eles descrevem as licenças como um conjunto de direitos com obrigações correspondentes e analisam como duas licenças interagem no contexto de cinco tipos de conexão entre componentes. A partir daí, na Seção 4 do artigo, eles apresentam doze padrões para evitar incompatibilidade de licenças, criados a partir de um estudo empírico com 124 projetos de software livre.

Também vale destacar o artigo *Dangerous Liaisons: Software Combinations as Derivative Works?* (2006) [Det06], em que Determann simula alguns casos de disputa legal, baseados em casos reais da jurisprudência americana, realizando uma análise jurídica profunda para determinar em quais situações um sistema deveria ser considerado como derivado de outro, afetando os direitos exclusivos devido a *copyright*.

Por fim, temos o capítulo *Software Livre e Propriedade Intelectual: Aspectos Jurídicos, Licenças e Modelos de Negócio* (2011) [KLMS11], cuja seção sobre licenças é baseada nesta dissertação, que contextualiza os aspectos jurídicos do software livre na forma como ele é utilizado, ressaltando as oportunidades e dificuldades que esse modelo representa.

Considerando os trabalhos relacionados que foram apresentados nesta seção, esta dissertação complementa os estudos citados ao apresentar as características das principais licenças em maiores detalhes e mostrar de que forma essas características têm impacto na compatibilidade entre licenças, além de contextualizar o estudo dentro da legislação brasileira.

Capítulo 2

Breve Histórico do Software Livre

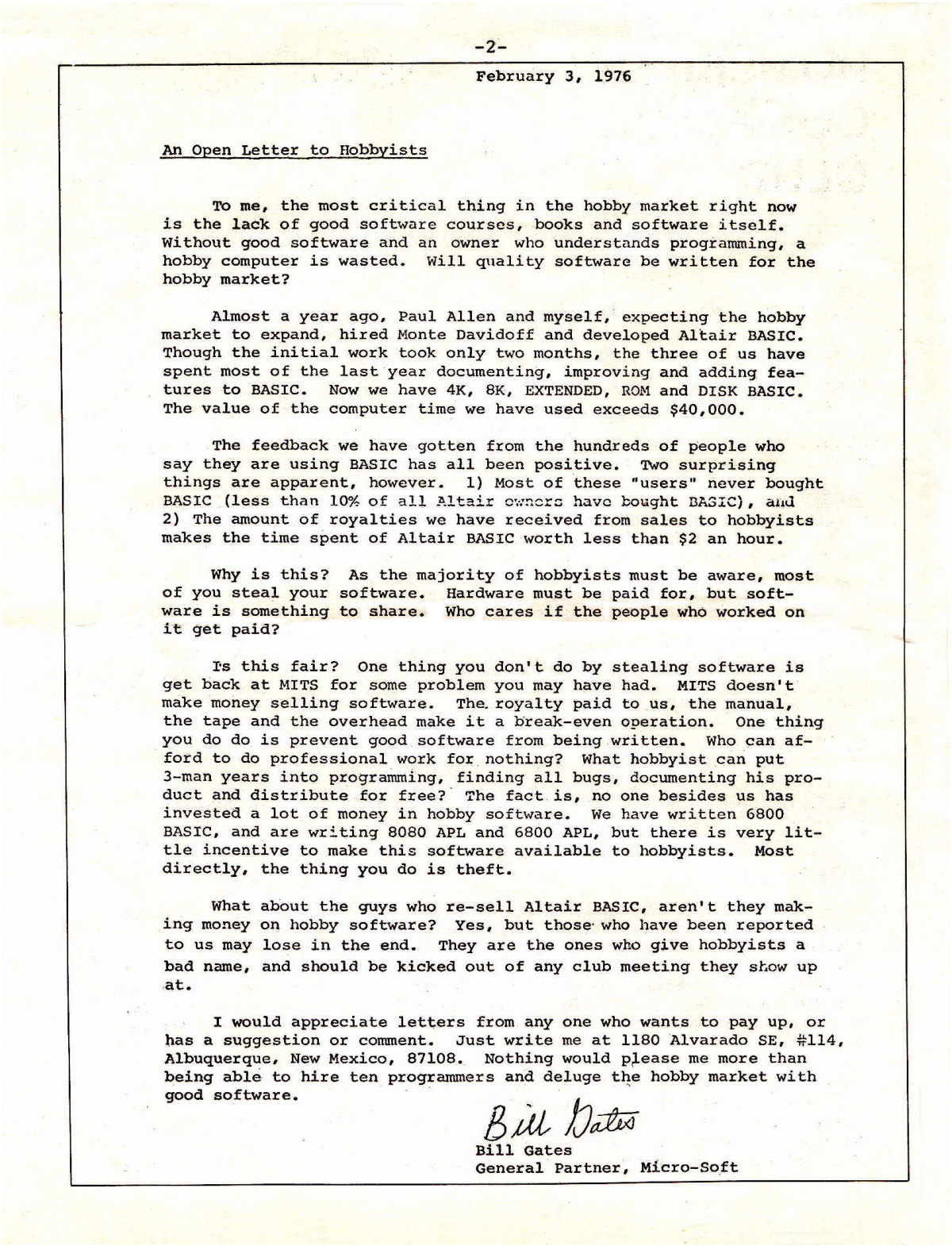
Com o surgimento dos primeiros computadores vendidos comercialmente, a partir da década de 1950, foram criados também os primeiros programas que iriam ser executados neles. Muitas vezes ocorria uma venda casada entre hardware e software, pois os programas eram fortemente acoplados à arquitetura das máquinas em que eram executados [Was11]. Nessa época, o foco das empresas era na venda do hardware, e não eram colocadas muitas restrições no uso que as pessoas fariam do software [CK08]. Elas podiam adaptá-lo como quisessem, de forma a fazer melhor uso do harware que tinham disponível, sem sofrer repreensões.

Na década de 1970 a situação começou a se modificar. Entre 1968 e 1969 a IBM anunciou o *unbundling* de seus produtos, separando o software do hardware, e já haviam empresas que focavam no desenvolvimento de software. Algumas dessas empresas, como a Microsoft, não estavam satisfeitas com a forma como seus programas eram muitas vezes redistribuídos sem que a empresa recebesse *royalties* pelas cópias. Assim, em 3 de fevereiro de 1976, Bill Gates escreveu a *Open Letter to Hobbyists*, que foi publicada na *newsletter* do *Homebrew Computer Club* (Figura 2.1). Nessa carta, Bill Gates afirma que o total de *royalties* recebidos pelo Altair BASIC era equivalente a apenas dois dólares por hora gasta em seu desenvolvimento e documentação. Ele ainda alega que a prática de compartilhamento de software não é justa e afirma que tal prática evita que software bem feito seja escrito. Assim, nessa época, começou uma mudança de postura na indústria, que passou a proibir que o software fosse copiado ou modificado. Surgiu então o que chamaremos de *software fechado*, caracterizado pelas restrições que são feitas à forma como ele será utilizado.

Como resposta a essa nova situação, surgiram iniciativas voltadas para retomar a liberdade de melhorar e compartilhar o software. Discutiremos a seguir duas delas: o Projeto GNU, combinado com a filosofia do Software Livre, e o movimento *Open Source* [Tau04].

2.1 O Projeto GNU e o Software Livre

Em 27 de setembro de 1983, Richard Stallman postou uma mensagem nos *newsgroups* `net.unix-wizards` e `net.usoft` com o assunto "*new Unix implementation*" [Sta83].



-2-

February 3, 1976

An Open Letter to Hobbyists

To me, the most critical thing in the hobby market right now is the lack of good software courses, books and software itself. Without good software and an owner who understands programming, a hobby computer is wasted. Will quality software be written for the hobby market?

Almost a year ago, Paul Allen and myself, expecting the hobby market to expand, hired Monte Davidoff and developed Altair BASIC. Though the initial work took only two months, the three of us have spent most of the last year documenting, improving and adding features to BASIC. Now we have 4K, 8K, EXTENDED, ROM and DISK BASIC. The value of the computer time we have used exceeds $40,000.

The feedback we have gotten from the hundreds of people who say they are using BASIC has all been positive. Two surprising things are apparent, however. 1) Most of these "users" never bought BASIC (less than 10% of all Altair owners have bought BASIC), and 2) The amount of royalties we have received from sales to hobbyists makes the time spent of Altair BASIC worth less than $2 an hour.

Why is this? As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?

Is this fair? One thing you don't do by stealing software is get back at MITS for some problem you may have had. MITS doesn't make money selling software. The royalty paid to us, the manual, the tape and the overhead make it a break-even operation. One thing you do do is prevent good software from being written. Who can afford to do professional work for nothing? What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free? The fact is, no one besides us has invested a lot of money in hobby software. We have written 6800 BASIC, and are writing 8080 APL and 6800 APL, but there is very little incentive to make this software available to hobbyists. Most directly, the thing you do is theft.

What about the guys who re-sell Altair BASIC, aren't they making money on hobby software? Yes, but those who have been reported to us may lose in the end. They are the ones who give hobbyists a bad name, and should be kicked out of any club meeting they show up at.

I would appreciate letters from any one who wants to pay up, or has a suggestion or comment. Just write me at 1180 Alvarado SE, #114, Albuquerque, New Mexico, 87108. Nothing would please me more than being able to hire ten programmers and deluge the hobby market with good software.

Bill Gates

Bill Gates
General Partner, Micro-Soft

Figura 2.1: *Homebrew Computer Club Newsletter Volume 2, Issue 1*

Nessa mensagem, ele informa que está começando a escrever um sistema compatível com UNIX[1] chamado GNU (um acrônimo recursivo para *Gnu's Not Unix*) e que ele será dado a todas as pessoas interessadas. Ele cita alguns componentes que seriam incluídos, tais como núcleo do sistema operacional, compilador C e editor de texto, e propõe algumas melhorias em relação aos sistemas UNIX existentes na época. Ele também explica na mensagem o motivo dele precisar escrever o GNU: segundo seus princípios, se ele gosta de um programa, então precisa compartilhá-lo com outras pessoas que também gostem dele. Para continuar usando computadores sem violar seus princípios, ele decidiu criar um conjunto suficiente de software livre para que ele pudesse prosseguir sem usar qualquer software que não fosse livre. Para finalizar a mensagem, ele pede contribuições na forma de máquinas, ajuda para escrever o software e dinheiro. No livro *Open Sources* [DOS99], Stallman conta um pouco mais sobre a história do projeto, cujos principais fatos serão discutidos aqui.

No início de 1984, Stallman pediu demissão do seu emprego no laboratório de Inteligência Artificial do MIT, para garantir que ele teria os direitos sobre o trabalho, e começou a dedicar-se em tempo integral ao projeto. O primeiro programa criado foi o GNU Emacs, que foi disponibilizado por FTP. Porém, como naquela época muitas pessoas não tinham acesso à Internet, Stallman também começou a ganhar dinheiro vendendo cópias físicas do programa, em fita magnética, por 150 dólares. Ele considera esse negócio de distribuição de software livre um precursor das empresas que hoje distribuem o GNU/Linux. Pouco tempo depois foi desenvolvido o GCC (que na época significava *GNU C Compiler*, mas atualmente, com a adição de outras linguagens, é chamado de *GNU Compiler Collection*), que até hoje é um dos componentes mais importantes do sistema GNU. Ao longo dos anos, Stallman decidiu incorporar ao sistema software que não foi escrito pelo projeto GNU, como por exemplo o Linux e o X Window System. Isso era possível pois esses programas também eram livres.

Enquanto o sistema GNU era desenvolvido, também foi sendo formado o conceito de *Free Software*, ou Software Livre, levando à criação da *Free Software Foundation* (FSF) por Stallman em 1985. Segundo sua definição, um software é livre se o usuário tem as seguintes liberdades:

1. A liberdade de executar o programa, para qualquer propósito.
2. A liberdade de modificar o programa para adaptá-lo às suas necessidades (para tornar essa liberdade efetiva na prática, é necessário ter acesso ao código fonte, já que fazer alterações em um programa sem ter o código fonte é muito difícil).
3. A liberdade de redistribuir cópias gratuitamente ou mediante pagamento.
4. A liberdade de distribuir versões modificadas do programa para que a comunidade possa se beneficiar de suas melhorias.

[1] Sistema operacional multi-tarefas e multi-usuários criado em 1969 por um grupo da AT&T.

Como o objetivo do projeto GNU era garantir essas liberdades para os usuários, foi criado um sistema de distribuição chamado *copyleft*, que buscava impedir que o software se tornasse fechado. Mais detalhes a respeito disso serão vistos na Seção 5.2, onde serão discutidas as licenças GPL.

2.2 O Movimento *Open Source*

Segundo Eric Raymond, devido ao caráter ético e político do movimento proposto pela *Free Software Foundation*, muitas empresas viam o software livre como "anti-capitalista" e eram relutantes em adotá-lo. Observando isso, ele teve a ideia de mudar a abordagem de como seria apresentado o software livre para pessoas e empresas mais conservadoras, e criou o termo *Open Source* em 1997. Ao não usar a palavra "*free*", Raymond estava não apenas evitando a confusão com gratuito, mas também tirando a conotação esquerdista e libertária do termo proposto por Stallman.

Também em 1997, Bruce Perens havia escrito o *Debian Free Software Guidelines*, para definir o que seria aceito como software livre pela distribuição Debian do GNU/Linux, dado que havia outras licenças além daquelas propostas pela *Free Software Foundation* que alegavam serem livres. Então, Eric Raymond contatou Bruce Perens para discutir a ideia de *Open Source* e decidiram, a partir daí, adaptar o documento *Debian Free Software Guidelines* para formar a *Open Source Definition*, ou Definição de Código Aberto. Eles registraram a marca *Open Source* e formaram a *Open Source Initiative* (OSI) [DOS99].

No início de 1998 aconteceu o primeiro caso de uma empresa já consolidada no mercado abrir o código de seu software, o Netscape. Junto com Eric Raymond, os executivos da Netscape escreveram uma licença que adotava alguns princípios de *copyleft*, mas que permitia à Netscape continuar distribuindo versões fechadas em paralelo com o projeto de código aberto. Em seguida, Raymond publicou um pedido à comunidade intitulado *Goodbye, "Free Software"; Hello, "Open Source"*, em que insistia que o termo *open source* era melhor do que *free software* e devia ser adotado.

Com essa nova abordagem, que ressaltava os benefícios técnicos decorrentes da metodologia adotada pela comunidade, e tendo como exemplo a Netscape, a adoção do software livre por parte das empresas sofreu um grande impulso. Porém, a maior disseminação do software livre nesses termos não deixou Stallman satisfeito. Segundo ele, se as pessoas não introjetarem a liberdade associada ao software livre, elas voltarão a usar software fechado quando este apresentar vantagens práticas.

Stallman também argumenta que a expressão "código aberto" tem como significado óbvio simplesmente "você pode olhar para o código", o que é um critério muito mais fraco do que a definição oficial de código aberto, que pode ser vista a seguir.

2.2.1 Definição de Código Aberto

Introdução

Código aberto não significa apenas acesso ao código fonte. Os termos de distribuição do software de código aberto devem estar de acordo com os seguintes critérios (tradução livre da *Open Source Definition* publicada em www.opensource.org/docs/osd):

1. Redistribuição Livre

A licença não deve restringir qualquer das partes de vender ou doar o software como um componente de uma distribuição agregada de software, contendo programas oriundos de várias fontes diferentes. A licença não deve exigir *royalties* ou qualquer outro tipo de pagamento pela exploração feita por terceiros.

2. Código Fonte

O programa deve incluir o código fonte e deve permitir a distribuição na forma de código fonte, bem como na forma compilada. Quando alguma forma do produto não é distribuída com o código fonte, é necessário haver meios bem divulgados para obtenção do código por não mais que um custo razoável de reprodução, preferencialmente através de *download* pela Internet gratuitamente. O código fonte deve ser a forma preferencial pela qual um programador alterará o programa. Código fonte ofuscado deliberadamente não é permitido. Formas intermediárias, como a saída de um processador ou tradutor, não são permitidas.

3. Trabalhos Derivados

A licença deve permitir modificações e trabalhos derivados e precisa permitir que eles sejam distribuídos sob os mesmos termos da licença do software original.

4. Integridade do Código Fonte do Autor

A licença pode restringir a distribuição de código fonte em forma modificada somente se a licença permitir a distribuição de "arquivos de *patch*"[2] com o código fonte para o propósito de modificar o programa em tempo de compilação. A licença deve permitir explicitamente a distribuição do software compilado a partir de um código modificado. A licença pode exigir que trabalhos derivados usem um nome ou número de versão diferentes do original.

5. Sem Discriminação a Pessoas ou Grupos

A licença não deve discriminar qualquer pessoa ou grupo de pessoas.

6. Sem Discriminação a Áreas de Empreendimento

A licença não deve restringir qualquer pessoa a fazer uso do programa em uma área de empreendimento específica. Por exemplo, ela não pode restringir o uso do programa comercialmente ou o uso em pesquisas genéticas.

[2] *Patch* é o nome dado a arquivos texto que mostram as diferenças entre duas versões do código fonte, permitindo que uma pessoa que tenha a versão anterior atualize-a para a nova versão de forma automatizada.

7. Distruibuição da Licença

Os direitos associados ao programa devem ser gozados por todos para quem o programa é redistribuído, sem a necessidade de execução de licenças adicionais para essas partes.

8. A Licença Não Deve Ser Específica a um Produto

Os direitos associados ao programa não devem depender dele ser parte de uma distribuição específica de software. Caso o programa seja extraído dessa distribuição e usado ou distribuído nos termos da licença do programa, todas as partes para as quais o programa é redistribuído devem ter os mesmos direitos que aqueles concedidos em conjunto com a distribuição de software original.

9. A Licença Não Deve Restringir Outro Software

A licença não deve colocar restrições em outro software que seja distribuído junto com o software licenciado. Por exemplo, a licença não deve exigir que todos outros programas distribuídos no mesmo meio sejam software de código aberto.

10. A Licença Deve Ser Neutra às Tecnologias

Nenhuma condição da licença deve ser estabelecida em uma tecnologia individual específica ou estilo de interface.

Baseando-se na Definição de Código Aberto, a OSI aprova as licenças que podem ser consideradas *open source*. Para isso, as licenças passam por um processo público de revisão para assegurar que estão em conformidade com as normas e expectativas da comunidade. A OSI conta com uma lista de mais de 70 licenças aprovadas (www.opensource.org/licenses) e possui um comitê para tratar do assunto de proliferação das licenças (que será apropriadamente elaborado no Capítulo 5), com o objetivo de criar mecanismos para facilitar a escolha da licença.

Capítulo 3

A Importância do Software Livre

Estima-se que hoje haja centenas de milhões de usuários de software livre no mundo[1]. Se considerarmos também usuários indiretos, que usam serviços baseados em software livre, tais como o Google, esse número provavelmente ultrapassa 1 bilhão de usuários. Neste capítulo, veremos as principais vantagens e desvantagens deste modelo, além das perspectivas de longo prazo.

3.1 Vantagens do Software Livre

Apesar da *Free Software Foundation* e da *Open Source Initiative* apresentarem justificativas diferentes para o uso do software livre, nesta dissertação nos concentramos nos pontos em comum entre as duas abordagens.

A principal vantagem do software livre é permitir o compartilhamento do código fonte. Como consequência desse compartilhamento, evita-se a duplicação de esforços quando mais de uma entidade está interessada no desenvolvimento de uma aplicação com funcionalidade similares, reduzindo assim o custo do desenvolvimento. Outro dividendo positivo é que o conhecimento contido no software passa a ser disponibilizado a todos.

Além disso, autores como Eric Raymond [Ray01] afirmam que software livre tem condições de ter maior qualidade do que seus equivalentes fechados. Uma das justificativas para essa afirmação de Raymond é conhecida como "A Lei de Linus", que diz que "dados olhos suficientes, todos os *bugs* são superficiais". Isso significa que, com o maior número de usuários que tem acesso ao programa e seu código fonte, o software é testado melhor e os problemas existentes no código são encontrados mais rapidamente. Outro fator que contribui para a qualidade é o orgulho pessoal do desenvolvedor pois, a partir do momento que seu código pode ser lido por mais pessoas, ele tende a ser mais cuidadoso com seu trabalho. A competição também é facilitada no software livre e, assim, se o grupo original de desenvolvedores não está fazendo um bom trabalho para manter o projeto, é possível que um novo grupo faça um *fork* para suprir as deficiências.

[1] Estimativa baseada no *market-share* do navegador Firefox (vide blog.mozilla.com/metrics/2010/03/31/mozillas-q1-2010-analyst-report-state-of-the-internet e no número de usuários de Internet (vide www.internetworldstats.com/stats.htm).

No Brasil, caracterizado por um forte mercado local, onde apenas uma pequena parte da indústria de software usa o modelo de desenvolvimento de "software de prateleira" (venda do produto sem customizações), o software livre pode trazer ainda mais benefícios. Ao desenvolver serviços e soluções baseadas em software livre, as empresas brasileiras podem deixar de investir no pagamento de licenças para software produzido em outro país. Dessa forma, é possível evitar a evasão de divisas, que contribui negativamente para a balança comercial brasileira.

Para os usuários também é vantajoso o software livre, pois evita a dependência de um fornecedor. Isso traz tanto uma vantagem financeira, dado que normalmente é necessário pagar por novas versões do sistema quando o software é fechado, como também maior liberdade para o usuário, que pode adaptar o software para suas necessidades. É possível corrigir falhas de segurança e *bugs*, escrever uma documentação melhor ou contratar uma empresa que faça isso independentemente de quem seja o autor original [Web04]. Além disso, se o fornecedor original abandona o projeto, no caso do software fechado não há nada que possa ser feito para continuar o desenvolvimento do projeto, enquanto que no software livre é possível que outro grupo adote o projeto e continue a evoluir o código.

3.2 Desvantagens do Software Livre

Também podemos levantar algumas desvantagens do software livre em relação a alternativas fechadas. Um dos principais motivos que leva uma empresa a optar por um software fechado quando há um similar livre disponível é a ausência de garantias e suporte desse último. As licenças de software livre em geral eximem o autor de qualquer responsabilidade tanto quanto é permitido pelas leis do local. Dessa forma, em casos em que a empresa precisa fornecer garantias aos seus clientes, ou quando a indisponibilidade de um sistema pode causar grandes prejuízos, pode ser melhor que a empresa adquira uma solução em que eventuais problemas sejam delegados a um fornecedor ou que esse tenha que indenizar a empresa. Porém, é importante deixar claro que, apesar das licenças de software livre normalmente incluírem cláusulas sobre a ausência de responsabilidades, há casos em que empresas optam por fornecer, como um serviço, garantias e suporte para um determinado software livre. Além disso, grande parte do software fechado disponível também busca em seu contrato se eximir de responsabilidades tanto quanto a legislação permite.

Qualidade, reputação e imagem também são vistos como desvantagens na adoção do software livre. Quando não há uma empresa de renome por trás do software oferecido, há uma maior dificuldade em avaliar as alternativas, além de um receio de que o produto seja abandonado e deixe-se de oferecer suporte para ele. Também influi negativamente na sua imagem o fato do software estar disponível gratuitamente.

Já do ponto de vista de quem produz software, optar pelo modelo aberto pode ser visto como desvantagem na medida em que a propriedade intelectual está exposta. Como os concorrentes têm fácil acesso ao código fonte, é necessário que a empresa criadora

do software mantenha seu diferencial para garantir seu mercado a longo prazo. Caso contrário, o software pode tornar-se um *commodity* e as possibilidades de lucro da empresa são reduzidas [Web04]. Por outro lado, no caso de uma licença que obriga que as melhorias sejam disponibilizadas como software livre, é mais difícil que uma nova empresa com um modelo de negócio que usa o mesmo software adquira vantagem competitiva sobre outra que já está consolidada no mercado, pois qualquer diferencial no produto pode ser prontamente absorvido pela empresa líder [DOS99].

3.3 Perspectivas do Software Livre

O modelo do software livre possui características bastante interessantes para o mercado de Tecnologia da Informação. Há uma certa ambiguidade na relação entre clientes, empresas e a comunidade de desenvolvimento de software, permitindo novos modelos de negócio. Os consumidores têm a possibilidade de também contribuir para o bem coletivo, agregando valor. Porém, a parcela de consumidores que realmente contribui é muito pequena, e os demais requerem uma quantidade desproporcional de serviços de suporte, que não são facilmente escaláveis. A competição com produtos fechados, que já arrecadam dinheiro com a venda do software, empurra para baixo o valor do suporte, dificultando a situação das empresas que têm como modelo a venda de serviços baseados em software livre [Web04].

Em um estudo da Forrester [For08] sobre o uso de software livre e seu impacto na indústria de software, foram observadas as seguintes tendências:

- crescimento da adoção de software livre pelo usuário final, na forma de ferramentas de produtividade e aplicações de negócio;
- crescimento de provedores de serviço e centros de competência para prover suporte comercial para produtos de software livre;
- diferenças na adoção de software livre de acordo com ramo de atividade, com fábricas adotando fortemente o software livre em sua infraestrutura enquanto serviços financeiros usam o software livre em aplicações de mais alto nível;
- uso de software livre em aplicações de missão crítica, serviços e produtos;
- altos índices de satisfação em relação a custo e qualidade do software livre;
- diversidade de critérios como motivadores para adoção de software livre, destacando custo, independência, flexibilidade e inovação;
- adoção, por parte das empresas, das boas práticas e princípios da comunidade de software livre.

Hoje em dia o software livre aparece na maior parte das projeções sobre o futuro do software. Porém, é importante ressaltar que cada novo paradigma que surge na indús-

tria altera substancialmente as perspectivas, tornando muito difícil prever como estará o mercado de software daqui a alguns anos [CK08].

Capítulo 4

Aspectos Jurídicos que Embasam as Licenças de Software

O software pode ser visto como uma criação recente da humanidade se comparado a outros tipos de obras, tais como literatura e música. Seu surgimento levou à necessidade de um tratamento jurídico especial desse novo *fato* devido ao seu caráter único.

Uma das grandes dificuldades encontradas é o enquadramento do software em alguma das categorias jurídicas pré-existentes, como afirma Augusto Marcacini [MdC04]:

> "O software é algo que não guarda paralelo exato com bens materiais ou imateriais até então conhecidos. Não é propriamente uma expressão da personalidade humana, mas um conjunto de instruções técnicas que devem levar o computador a produzir um resultado desejado. Mas também não se aproxima, por exemplo, de um projeto para construir um avião a jato, pois o software é um conjunto de instruções que funciona e é útil por si, enquanto as plantas e desenhos de um avião não transportam nada nem ninguém para lugar algum. Seria, então, muito mais uma mercadoria, uma *res*, do que um projeto apenas; entretanto, na sua essência, o software é um conjunto de ideias que estão gravadas em meio eletrônico e podem ser infinitamente copiadas o que o distancia, pela imaterialidade, da noção de *res*, aproximando-se, neste aspecto prático, dos direitos de natureza autoral, de uma criação intelectual, e aqui voltamos para o começo do parágrafo, numa espiral infinita."

No final da década de 1970, um grupo conhecido como *Advisory Group of Governamental Experts on the Protection of Computer Programs* levantou alguns dos motivos pelos quais a proteção do software era desejável [dS08]:

- investimento e tempo necessários para desenvolvimento de software;
- probabilidade de futuros desenvolvimentos;
- necessidade de se criar um incentivo para a divulgação;
- utilização da proteção como base para o comércio;

- vulnerabilidade do software.

O direito prevê diversas formas de proteção da solução técnica que faz parte do software, considerada como uma propriedade imaterial [dS08]. As principais delas são:

- patentes;
- direito autoral;
- segredo comercial e industrial;
- desenho industrial;
- marcas;
- repressão à concorrência desleal.

Para entendermos as leis relacionadas ao software, alguns pontos valem ser discutidos. O primeiro é a diferença entre software e *programa de computador*. O programa de computador está definido no primeiro artigo da Lei nº 9.609/98: "Programa de computador é a expressão de um conjunto organizado de instruções em linguagem natural ou codificada, contida em suporte físico de qualquer natureza, de emprego necessário em máquinas automáticas de tratamento da informação, dispositivos, instrumentos ou equipamentos periféricos, baseados em técnica digital ou análoga, para fazê-los funcionar de modo e para fins determinados". Já a definição de software não é muito precisa, mas engloba outros elementos que estão relacionados ao programa, tais como sua documentação (tanto a técnica como a destinada ao usuário final) [Bar03]. As leis 7.232/84 e 9.609/98 tratam esses conceitos de forma distinta: a primeira trabalha estes elementos e o programa de computador como uma só entidade, o *software*, cuja importação, exportação, produção, operação e comercialização são definidos como atividades sujeitas a ela; enquanto que a última define e regula o programa de computador e só cita os elementos acessórios quando eles se fazem necessários como, por exemplo, o código fonte e documentação na ocorrência de transferência de tecnologia.

Outro ponto importante é a diferença entre código fonte e código binário. Há consenso em relação ao código fonte constituir obra literária na acepção geral adotada pelo Direito do Autor [dS08]. Já o código binário constitui uma evolução substancial da versão em código fonte e não se pode afirmar diretamente que a cópia ou reprodução do binário é indiretamente uma cópia do código fonte [Cor95]. Há razões técnicas que tornam não razoável uma análise de *copyright* de código binário, pois ele é na maioria das vezes produzido pelo próprio computador e não é legível para humanos.

Enquanto os usuários adotam uma visão lógica que diferencia apenas um software instalado de um software em execução, durante o funcionamento de um programa de computador ocorrem diversas atividades na camada física que viabilizam esse processo. Considerando um computador convencional, as instruções que definem o programa, quando não estão em uso, residem normalmente no disco rígido ou alguma outra mídia não volátil,

como um CD-ROM ou pendrive. A partir do momento que esse programa é executado, essas instruções precisam ser copiadas para a memória RAM e, no momento em que estão em uso, vão também para o cache do processador, que pode ser único ou estar dividido em diversos níveis (em geral L1 e L2). O cache é a memória de acesso mais rápido, seguido pela RAM, que é muitas vezes mais rápida que um disco rígido. Por outro lado, as memórias mais rápidas são também muito mais caras e por isso estão disponíveis em menor quantidade no computador. Isso significa que não seria viável ter um programa inteiro copiado no cache, e, dependendo de seu tamanho e de quantas tarefas estão sendo executadas simultaneamente, não é possível nem mesmo ter todas as instruções e dados sendo usados pelo computador ao mesmo tempo na memória RAM. Por esse motivo, há a necessidade de copiar os *bits* constantemente de uma memória para outra. O arranjo físico desses *bits* é gerenciado em grande parte pelo sistema operacional e está baseado principalmente em questões técnicas de eficiência de armazenamento, tendo pouca relação com o funcionamento lógico do programa. Como essas atividades são ditadas por requisitos de funcionalidade externa (armazenamento e eficiência de execução) e não têm ligação a aplicações de software específicas e a expressão de criatividade contida nelas, esses eventos de cópia de trechos e combinação que ocorre no nível físico são considerados irrelevantes para fins de análise dentro da lei de *copyright*. Ao invés disso, o foco é na perspectiva lógica, que vê programas como trabalhos separados baseando-se em como eles são escritos por seus autores e percebidos pelos usuários [Det06].

Outro fator a ser considerado é que o software algumas vezes é visto como prestação de serviços. Tivemos, por exemplo, um caso que chegou ao Supremo Tribunal Federal para resolver a ambiguidade entre mercadoria e serviço do ponto de vista da tributação do software. A decisão do tribunal determinou que "se o contrato é de cessão ou envolve desenvolvimento de produto para o atendimento de determinada necessidade do contratante, caracteriza-se a prestação de serviços, sujeito ao pagamento do ISS; se consiste em licença para aquisição de direitos sobre software comercializado em larga escala e de maneira uniforme, há circulação de mercadorias, incidindo o ICMS" [dAMJ05].

Na prática, os programas de computador são regidos judicialmente no Brasil da seguinte forma: em primeiro lugar vale a Lei 9.609/98, conhecida como Lei do Software; em casos em que essa é omissa, aplica-se a Lei 9.610/98, do Direito Autoral; e sendo essa última ainda insuficiente, recorre-se ao Código Civil [FJL+05].

É importante ressaltar, contudo, que ainda há muito pouco trabalho sobre licenciamento de software livre por parte dos juristas brasileiros. Também há poucos casos de jurisprudência. Segundo Marcacini [Mar10], é muito difícil julgar qualquer causa sem compreender o *fato* do qual decorre o direito das partes, e para muitos juristas faltam conhecimentos específicos sobre as questões que envolvem o software. Portanto, a doutrina ainda precisa ser melhor desenvolvida, para fortalecer a teoria jurídica sobre esse assunto.

4.1 Histórico

No início da década de 1980, surgiam no Brasil os primeiros estudos sobre proteção do software. Segundo Marcacini [Mar10], "se atuar em um caso concreto já se mostra tarefa difícil, mais complexa ainda é a situação do estudioso de Direito quando se depara com a necessidade de propor novas leis sobre esses fatos tecnológicos, ou construir bases doutrinárias a seu respeito". Fatores tais como o benefício da sociedade no longo prazo e a necessidade de compatibilização com a legislação que já cobria outras criações tecnológicas foram levantados em um relatório do escritório brasileiro de patentes em 1983 [dS88], com a proposta de guiar as decisões que seriam tomadas a seguir. Em 1984, foi apresentado um anteprojeto de lei (260/84), redigido pela Secretaria Especial de Informática (SEI), Instituto Nacional da Propriedade Industrial (INPI) e Conselho Nacional de Direito Autoral (CNDA), que propunha proteção mediante registro do software. Nesse projeto, já estava previsto que semelhanças decorrentes da característica dos equipamentos, observância de padrões técnicos ou normas legais, emprego de algoritmos em domínio público, modelos matemáticos ou processos formais não constituiria violação dos direitos exclusivos concedidos, desde que se tratasse de uma criação independente [dS08]. No final da década de 1980, através da Lei nº 7.646/87 (posteriormente revogada pela Lei nº 9.609/98), foi escolhido o direito autoral como regime de proteção do software, alinhando-se à estratégia adotada nos Estados Unidos.

Não foi apenas o Brasil que estava discutindo a legislação para proteção do software nessa época. Desde a década de 1970, a OMPI (Organização Mundial da Propriedade Intelectual) promoveu estudos a esse respeito, escrevendo um documento intitulado "Disposições Tipo para a Proteção do Software" em que sugeria um regime jurídico especial para esse objeto. Porém, na década de 1980 vários países produtores de software começaram a adotar o direito autoral, justificado pela necessidade de encontrar uma proteção rápida e eficaz que fosse aceitável no plano internacional [dS08]. Apesar do direito autoral ter sido adotado amplamente, a maioria dos países adotou também regras específicas para lidar com software, diferenciando-o de outras obras intelectuais. Na União Européia, por exemplo, a Diretiva 91/250/CEE, de 1991, aproximou o direito autoral europeu (baseado no conceito francês de *droit d'auteur*) do direito da *common law* (onde temos o *copyright*) adotado nos países anglo-saxônicos, principalmente no quesito originalidade.

Na década de 1990, surgiu também o acordo TRIPS (*Trade Related Aspects of Intellectual Property Rights*), que deveria ser seguido por todos os países membros da OMC. Um dos dispositivos desse acordo estabelecia que programas de computador, em código fonte ou objeto, são protegidos como obra literária pela Convenção de Berna. Até então, vários países consideravam os programas como obra de arte aplicada, cuja proteção é inferior, inclusive em relação à duração de tempo em que a obra fica protegida.

4.2 Direito Autoral e a Lei do Software

Segundo Ronaldo Lemos em Falcão et al. [FJL+05],

> "O licenciamento em software livre nada mais é do que uma modalidade de exercício dos direitos do autor através de uma licença jurídica".

As leis relacionadas à proteção autoral devem buscar balancear os interesses do detentor dos direitos com os do público, beneficiando toda a sociedade no longo prazo [Ben06]. No caso de programas de computador, cuja finalidade muitas vezes é meramente utilitária, a funcionalidade e valor econômico costumam ser mais importantes do que um valor cultural associado e, assim, para estabelecer equilíbrio, as leis devem buscar maior liberdade de concorrência.

O direito autoral cobre formas de expressão que são objetos concretos, e não ideias. Ou seja, ele está protegendo a implementação específica que aparece no código fonte, mas não a funcionalidade do software, que é algo mais abstrato. Na proposta da lei autoral australiana de 1984, por exemplo, havia um comentário para esclarecer essa diferença em sua definição de programa de computador, afirmando: "*'expression ... of a set of instructions' is intended to make it clear that it is not an abstract idea, algorithm or mathematical principle which is protected, but rather a particular expression of that abstraction*". Assim, o direito de exclusividade do autor é exercido essencialmente sobre o código do programa, daí seu enquadramento como obra literária [dS08]. Da mesma forma que o algoritmo não é protegido pelo direito autoral, estruturas de dados também não o são. Ideias normalmente são cobertas por patentes, porém, pela lei brasileira (9.279/96), não se considera um programa de computador como invenção ou modelo de utilidade. O software, como conjunto específico de instruções, é protegido automaticamente pelo direito autoral, sendo seu registro perante o INPI facultativo. Através do direito autoral é impedida a mera duplicação da codificação, o que permite coibir legalmente a pirataria de programas de computador.

Apesar de ser reconhecida a proteção do programa de computador sob o direito autoral, a legislação inclui ressalvas que diferenciam software de outras obras. A Lei nº 9.609/98 trata dessas diferenças, a principal delas estando no inciso 1 do artigo 2º, que afirma que "não se aplicam ao programa de computador as disposições relativas aos direitos morais". Dessa forma, direitos relacionados à personalidade do autor, tais como retirar a obra do mercado ou proibir usos que já haviam sido autorizados, não valem para software, porém outros direitos, tais como reivindicar a autoria e vetar reproduções, continuam válidos.

Uma dificuldade encontrada na interpretação de licenças de software livre no Brasil é que a maioria delas foi escrita tendo como base a lei de *copyright* americana, que determina direitos exclusivos apenas em casos mais específicos, incluindo o direito de copiar, distribuir e adaptar o trabalho. Na legislação brasileira isso funciona de forma diferente: qualquer tipo de uso tem que estar expressamente autorizado. O artigo 29º da

Lei nº 9.610/98 lista diversos casos de uso como sendo protegidos e finaliza com "quaisquer outras modalidades de utilização existentes ou que venham a ser inventadas".

A atual lei de direito autoral lista, no artigo 8º, os elementos que não são objeto de proteção como direitos autorais, entre os quais estão ideias, procedimentos normativos, sistemas, métodos, esquemas, planos etc. Por outro lado, há elementos não literais de uma obra que podem ser protegidos pelo direito autoral. Nos Estados Unidos, há alguns casos de jurisprudência em que semelhanças em um desses elementos não-literais, no caso o *look and feel*[1] de aplicativos, levou as companhias envolvidas aos tribunais com reivindicações baseadas nos direitos autorais. Observou-se que o argumento geral das grandes empresas era a favor de ampliar a proteção dos direitos autorais, enquanto cientistas da computação, juristas e grupos de usuários sustentavam a posição contrária. Tal fato evidencia que o direito de autor "tem servido muito mais ao interesse comercial das grandes empresas do que ao progresso da ciência ou ao interesse dos usuários, comprometendo assim o delicado equilíbrio entre o interesse público e o privado" [dS08]. Os casos mais famosos nesse sentido foram da Apple vs. Microsoft [Uni94], em 1995, e Lotus vs. Borland [Uni96], em 1996. Em ambos a decisão final da corte foi em favor do réu, não repreendendo o uso de conceitos similares na interface do usuário.

A aplicação do conceito de elementos não-literais a programas de computador não é direta, porém, em geral, está relacionada à estrutura e organização do mesmo. Por exemplo, pela Lei nº 9.610/98, artigo 87º, o titular do direito patrimonial sobre uma base de dados terá direito exclusivo a respeito da forma de expressão da estrutura da referida base. Por outro lado, a Lei de Software brasileira não trata da estrutura de programas, e ainda dispõe que a semelhança entre dois programas decorrente das características funcionais da aplicação dos mesmos não constitui infração [dS08]. Um programa é cópia de outro quando implementa os procedimentos lógicos do programa preexistente de maneira substancialmente similar [Fri88]. Pesquisadores como José de Oliveira Ascensão ainda afirmam que para haver originalidade no modo de expressão, do ponto de vista autoral, é necessário que haja mais de uma alternativa para implementar determinado processo [dOA97]. Porém, a forma de implementação de determinada funcionalidade nem sempre é evidente e, assim, muitas vezes é necessário criatividade e originalidade mesmo quando não há formas alternativas.

Outro ponto de nossa legislação que vale ser ressaltado é a proteção da interface lógica dos programas. No capítulo da Lei do Software relativo às limitações aos direitos de autor, contempla-se a hipótese de integração de um programa a um sistema aplicativo ou operacional, tecnicamente indispensável às necessidades do usuário, desde que, como dispõem o Art. 7º, IV, da Lei nº 7.646/87 e o Art. 6º da Lei nº 9.609/98, mantendo suas características essenciais e para uso exclusivo de quem a promoveu. Dessa forma, foram contempladas as exigências de interoperabilidade de dois programas que determinam certas restrições aos direitos exclusivos do titular do direito de autor [dS08].

[1] Termo que denota características de *design* e comportamento de uma interface gráfica.

4.3 Contratos

A Lei nº 9.609/98 determina no Art. 9º que "o uso de programa de computador no País será objeto de contrato de licença". Ao contrário da cessão dos direitos patrimoniais, em que todos os direitos seriam transmitidos ao cessionário, através do uso de licenças é dada apenas uma autorização de uso.

Segundo a teoria contratual, o contrato é como "um acordo realizado pela livre vontade de partes formalmente iguais para realização de trocas de bens ou serviços mediada por um valor equalizante" [FJL+05]. Conforme afirma Augusto Marcacini, "a vontade livremente manifestada em um contrato também é considerada *lei* entre as partes, merecendo destaque que, no campo dos direitos patrimoniais e obrigacionais, a liberdade de contratação é regra: o que a lei não proíbe, o que não afronta a ordem pública, é permitido" [MdC04]. Caio Pereira ainda define contrato como "um acordo de vontades, na conformidade da lei, e com a finalidade de adquirir, resguardar, transferir, conservar, modificar ou extinguir direitos" [dSP09].

Os contratos podem ser divididos entre típicos e atípicos. Os contratos típicos são aqueles cujas regras disciplinares estão determinadas nas leis. Por exemplo, o Código Civil lista contratos como os de compra, venda e locação, além de outros. Os contratos que não estão regulamentados, envolvendo novas relações jurídicas, são chamados de atípicos. Como esses contratos não estão estabelecidos nas leis, é necessário detalhar todos os direitos e obrigações que os compõem. Na hipótese de um contrato desse tipo ser levado a julgamento, cabe ao juiz interpretar o contrato utilizando os princípios legais relativos ao contrato típico mais próximo e aos contratos em geral.

Quando o conteúdo de um contrato é determinado unilateralmente por um dos contratantes, cabendo à outra parte apenas aderir aos termos estabelecidos, o chamamos de contrato de adesão. O Código Civil determina, no artigo 423º, que eventuais cláusulas ambíguas ou contraditórias sejam interpretadas da maneira mais favorável à parte que não teve o poder de influir no conteúdo do contrato [dSP09].

Licenças de software livre são classificadas por alguns como contratos benéficos, também chamados de gratuitos ou desinteressados. Esse tipo de contrato é caracterizado por apenas uma das partes auferir os benefícios, enquanto a outra sofre os encargos. Em relação a licenças recíprocas (vide Capítulo 5), pode-se ainda afirmar que são contratos benéficos com estipulações em favor de terceiro [FJL+05], pois não é pré-determinado quem serão as demais pessoas que se beneficiarão da licença no momento em que o contrato é firmado entre duas partes. Essa classificação não é unânime por parte dos juristas e é importante ressaltar que uma licença de software livre não caracteriza uma abdicação de direitos.

Em licenças de software livre, o licenciante não escolhe como parte contratante apenas um indivíduo ou uma empresa. O contrato fica aberto a qualquer membro da comunidade. Alguns autores consideram as licenças de software livre recíprocas como *contratos de licenciamento em rede*:

> "Dizemos que é um contrato de licenciamento em rede porque institucionaliza uma livre reprodução de inovações e de uso do software em cadeia, através do mecanismo que faz com que o licenciado de hoje seja *ipso facto* o licenciante de amanhã. Num certo sentido, este contrato é uma espécie de contrato *viral*, na medida em que a cláusula do compartilhamento obrigatório inocula-se em todos os contratos, os fazendo partícipes de uma mesma situação" [FJL+05].

Nesse caso, o usuário que decide usar o software livre já compromete-se a compartilhar, sob a mesma licença, trabalhos futuros criados com base nesse software antes mesmo deles serem criados, exercendo, a priori, as decisões que a ele caberia pelo direito de autor. Isso, segundo os mesmos autores [FJL+05], pode parecer ir contra os fundamentos do direito e a liberdade de quem está recebendo o software, porém, não há restrições jurídicas que impeçam esse tipo de contrato. No software livre, considera-se que todas as partes são interdependentes, agindo ao mesmo tempo como contratantes e contratados. Mais ainda:

> "Esta cláusula de compartilhamento obrigatório exerce pelo menos duas funções. Primeiro, transforma o contrato de licenciamento numa oferta *erga omnes*[2], constituindo então uma rede aberta. Segundo, não estabelece nenhum impedimento para que, no futuro, este ou aquele licenciado entre na rede, sejam impedimentos com base em *status* jurídico, sejam com viés econômico, de sexo, raça, nacionalidade ou de qualquer outro tipo. Não cria nenhuma escassez legal, como no contrato liberal clássico. A única exigência necessária é o mero ato de adotar o software livre, por seu uso ou por sua recriação [FJL+05]".

Outra característica das licenças de software livre é a uniformidade contratual, dado que um mesmo tipo de contrato, com as mesmas cláusulas, é aplicado a todas as pessoas envolvidas. Assim, qualquer aperfeiçoamento do software que for disponibilizado pode beneficiar igualmente a todas as partes.

Sendo as licenças de software livre um negócio jurídico não-oneroso, a responsabilidade civil fica restrita às hipóteses em que houve dolo. Essa regra está no artigo 392º do Código Civil, que afirma: "nos contratos benéficos, responde por simples culpa o contratante, a quem o contrato proveite, e por dolo aquele a quem não favoreça. Nos contratos onerosos, responde cada uma das partes por culpa, salvo exceções previstas em lei". Segundo Marcacini, "a própria definição de fraqueza do consumidor, principal justificativa para a existência da legislação protetiva, inexiste aqui. Ninguém é obrigado ou compelido de qualquer forma a aceitar a licença e utilizar o software livre" [MdC04]. Por outro lado, nem sempre é caracterizada uma relação de consumo no software livre e muitas vezes uma organização encontra-se no meio da cadeia produtiva. O estudo da FGV ainda argumenta que uma relação de consumo pressupõe dois pólos de interesse, o consumidor e o fonecedor, enquanto licenças de software livre são sociais e colaborativas, envolvendo interesses de toda a coletividade, e não uma relação bipolar [FJL+05]. Porém, isso não

[2] Do latim "contra todos", isto é, com efeito legal sobre todos os envolvidos.

impede que outras relações jurídicas sejam estabelecidas a partir do software livre, sendo possível que essas relações incorram em outras obrigações e responsabilidades.

No próximo capítulo estudaremos em maiores detalhes as principais licenças de software livre.

Capítulo 5

Levantamento e Classificação de Licenças

O detentor dos direitos patrimoniais sobre um software, quando decide torná-lo livre, deve escolher os termos em que seu trabalho será distribuído, ou seja, os direitos que ele estará transferindo para as outras pessoas e quais as condições que serão aplicadas. O documento que formaliza esse ato é a licença, que normalmente é distribuída junto com o código fonte.

Há inúmeras possibilidades para redigir o texto de uma licença de software livre, mas a prática mais comum e recomendada é reaproveitar alguma das licenças já consolidadas na comunidade. Dessa forma, reduz-se a proliferação de licenças, que deve ser evitada pois gera trabalho adicional para os usuários, uma vez que torna-se necessário para eles estudar os termos de cada nova licença presente no software que irão utilizar. No portal SourceForge [Sou08] , que hospeda um grande número de projetos de software livre, ao criar um novo projeto são apresentadas oito opções de licenças, que são as mais utilizadas naquele repositório, além da opção de domínio público:

- GNU General Public License (GPL)
- GNU Library or Lesser General Public License (LGPL)
- BSD License
- MIT License
- Apache License V2.0
- Artistic License
- Mozilla Public License 1.1 (MPL 1.1)
- Academic Free License (AFL)

Além disso, no final da página, há um ponteiro para visualização de todas as licenças disponíveis, contendo, em maio de 2010, 64 opções. Há ainda a possibilidade do usuário incluir sua própria licença.

A *Open Source Initiative* (OSI) mantém uma lista de licenças aprovadas de acordo com a definição de código aberto (vide Seção 2.2), organizadas em diversas categorias, além de uma lista de licenças atualmente buscando aprovação. A OSI também está trabalhando com um comitê de proliferação de licenças, buscando separar e dar destaque a algumas licenças que passarão a ser *recomendadas*, em oposição às meramente *aprovadas*.

Vamos discutir neste capítulo algumas das principais licenças utilizadas atualmente pela comunidade, buscando seu melhor entendimento e aplicação. Considerando que uma das questões mais importantes é a compatibilidade entre licenças, que será discutida no Capítulo 6, vamos separar as licenças descritas aqui em três categorias, de acordo com a presença de termos que impõem restrições de licenciamento na redistribuição do trabalho ou criação de trabalhos derivados. Em relação a essa característica, as licenças são consideradas *permissivas* ou *recíprocas*, sendo que entre as recíprocas devemos ainda considerar que algumas forçam que seja mantida a mesma licença em mais casos do que outras e, assim, as dividimos entre recíprocas parciais, que também recebem a denominação de *copyleft* fraco, e recíprocas totais.

5.1 Permissivas

As licenças permissivas, também chamadas de licenças acadêmicas por alguns autores, como Rosen [Ros05] e Laurent [Lau04], em referência às origens das licenças BSD (*University of California, Berkeley*) e MIT (*Massachusetts Institute of Technology*), impõem poucas restrições às pessoas que obtém o produto. Muitas vezes, tais licenças são usadas em projetos de pesquisa de universidades, que servem como prova de conceito de alguma tecnologia que poderá ser explorada comercialmente no futuro. No caso das licenças permissivas, não são colocadas grandes restrições ao licenciamento de trabalhos derivados, estes podendo inclusive serem distribuídos sob uma licença fechada. Nesta seção serão discutidas as licenças BSD, MIT e Apache.

Licenças permissivas são uma ótima opção para projetos cujo objetivo é atingir o maior número de pessoas, não importando se na forma de software livre ou de software fechado. Temos vários exemplos desse modelo no BSD Unix, que continha o software de TCP/IP que hoje é usado na maior parte das implementações desse protocolo, incluindo a da Microsoft [Lau04]. Outro exemplo é o BIND(*Berkeley Internet Name Daemon*), cuja implementação livre é até hoje usada nos principais servidores de DNS, apesar de haver também várias implementações fechadas. Segundo Laurent [Lau04], há bilhões de dólares em atividade econômica associada apenas com a pilha de software para Internet originalmente liberada sob a licença BSD.

Alguns argumentam que o uso desse tipo de licença não incentiva o modelo de software livre, pois empresas se aproveitam da comunidade para desenvolver software que será fechado. Porém, quando são usadas licenças permissivas, em geral os autores estão cientes dessa possibilidade e não vêem isso como um problema. Um caso conhecido em que, de fato, os autores se arrependeram da licença que adotaram é o do Kerberos, desenvolvido

no MIT, que posteriormente foi adotado pela Microsoft, que desenvolveu extensões fechadas [Lau04]. Mas o mais provável, caso a licença não permitisse isso, seria que a Microsoft adotasse algum outro sistema de segurança e o Kerberos não se tornaria tão popular.

Por outro lado, em alguns casos, não é necessário que haja restrições na licença para garantir a continuidade do modelo de desenvolvimento aberto, como no exemplo do servidor Apache. Duas características explicam o domínio do servidor Web livre no mercado, segundo Laurent [Lau04]: a marca forte, cujo uso é protegido pela própria licença do Apache, e a importância da conformidade com os padrões em sistemas Web, o que evita a disseminação de extensões fechadas.

5.1.1 A Licença BSD

A primeira licença que iremos estudar é a BSD (www.opensource.org/licenses/bsd-license.php), que foi a primeira licença de software livre escrita e até hoje é uma das mais usadas. Criada originalmente pela Universidade da Califórnia em Berkeley para seu sistema operacional derivado do UNIX chamado *Berkeley Software Distribution*, a licença BSD é usada como modelo por uma ampla gama de software licenciado de modo permissivo.

Os principais motivos que levaram a licença BSD a ser tão difundida são a simplicidade de seu texto e o fato dela ter sido inicialmente adotada por um projeto amplamente disseminado, o que criou um ciclo virtuoso em que mais comunidades a adotaram, tornando-a ainda mais reconhecida.

Porém, originalmente, a licença BSD possuia uma cláusula que determinava que todo material de divulgação relacionado ao software precisava conter uma sentença que dizia "este produto inclui software desenvolvido pela Universidade da California, Berkeley e seus contribuidores". Tal cláusula, que ficou conhecida como "cláusula de propaganda da licença BSD", não causava muito problema quando era usada apenas no seu contexto original, porém, à medida que outras pessoas e organizações passaram a adotar também a licença BSD, essa cláusula era adaptada para conter outros nomes, e assim surgiam projetos que integravam diversos componentes e determinavam uma longa lista de frases a serem incluídas quando se falava sobre o projeto. Como consequência, em 1999, tal cláusula foi removida, originando a "Licença BSD Simplificada", também conhecida como "Nova Licença BSD" ou "Versão de 3 cláusulas da Licença BSD".

Assim, na versão atualmente utilizada da licença BSD, temos os seguintes termos, apresentados em tradução realizada pela Escola de Direito da Fundação Getúlio Vargas [FGV]:

```
Direitos autorais (C)  <ANO>, <TITULAR>

Todos os direitos reservados.

A redistribuição e o uso nas formas binária e código fonte, com ou sem
modificações, são permitidos contanto que as condições abaixo sejam
cumpridas:

- Redistribuições do código fonte devem conter o aviso de direitos
autorais acima, esta lista de condições e o aviso de isenção de
garantias subsequente.

- Redistribuições na forma binária devem reproduzir o aviso de direitos
autorais acima, esta lista de condições e o aviso de isenção de
garantias subsequente na documentação e/ou materiais fornecidos
com a distribuição.

- Nem o nome da <ORGANIZAÇÃO> nem o nome dos contribuidores
podem ser utilizados para endossar ou promover produtos derivados
deste software sem autorização prévia específica por escrito.
```

Apesar dessa licença utilizar uma linguagem leiga, cujos termos não necessariamente correspondem aos termos encontrados nas leis que visam proteger o autor do software, através deste documento o detentor dos direitos autorais permite que outras pessoas usem, modifiquem e distribuam o software. As únicas exigências são que o nome do autor original não seja utilizado em trabalhos derivados sem permissão, de forma a proteger sua reputação, dado que o autor pode não ter relação alguma com as modificações realizadas, e no caso de redistribuição do código fonte ou binário, modificado ou não, é necessário que seja mencionado o *copyright* original e os termos da licença.

É importante ressaltar que a exigência de reproduzir a lista de condições para redistribuição e o aviso legal de isenção de garantias não significa que o trabalho sendo redistribuído precisa estar sob a mesma licença. Além disso, é possível redistribuir o binário sem fornecer o código fonte. Basta que conste na documentação que foi utilizado o software em questão, e que ele foi licenciado nos termos descritos acima.

No final da licença há o aviso legal sobre garantias e responsabilidades apresentado a seguir, que visa proteger o autor de ser processado judicialmente por qualquer insatisfação causada em consequência do uso do software. Porém, tal termo está subordinado às leis locais, que no caso brasileiro, dependendo da relação estabelecida entre o fornecedor e o cliente, não permitem a distribuição de um software com ausência total de garantias.

```
ESTE SOFTWARE É FORNECIDO PELOS DETENTORES DE DIREITOS AUTORAIS E
CONTRIBUIDORES "COMO ESTÁ", ISENTO DE GARANTIAS EXPRESSAS OU TÁCITAS,
INCLUINDO, SEM LIMITAÇÃO, QUAISQUER GARANTIAS IMPLÍCITAS DE
COMERCIABILIDADE OU DE ADEQUAÇÃO A FINALIDADES ESPECÍFICAS. EM NENHUMA
HIPÓTESE OS TITULARES DE DIREITOS AUTORAIS E CONTRIBUIDORES SERÃO
RESPONSÁVEIS POR QUAISQUER DANOS, DIRETOS, INDIRETOS, INCIDENTAIS,
ESPECIAIS, EXEMPLARES OU CONSEQUENTES, (INCLUINDO, SEM LIMITAÇÃO,
FORNECIMENTO DE BENS OU SERVIÇOS SUBSTITUTOS, PERDA DE USO OU DADOS,
LUCROS CESSANTES, OU INTERRUPÇÃO DE ATIVIDADES), CAUSADOS POR QUAISQUER
MOTIVOS E SOB QUALQUER TEORIA DE RESPONSABILIDADE, SEJA
RESPONSABILIDADE CONTRATUAL, RESTRITA, ILÍCITO CIVIL, OU QUALQUER OUTRA,
COMO DECORRÊNCIA DE USO DESTE SOFTWARE, MESMO QUE HOUVESSEM SIDO
AVISADOS DA POSSIBILIDADE DE TAIS DANOS.
```

Por fim, há também versões da licença BSD que removem a terceira condição de redistribuição, relacionada ao autor não endossar trabalhos derivados, restando apenas as duas cláusulas que requerem a reprodução do *copyright* e das condições na redistribuição.

Vantagens e Desvantagens

Uma das principais desvantagens da licença BSD é a quantidade de variantes existentes, pois não fica imediatamente claro para o usuário sob quais termos um software está sendo licenciado quando a única informação que ele recebe é que está sob a licença BSD. Sempre é necessário verificar com mais cuidado, no texto da licença, se o software em questão está licenciado pela BSD original, a simplificada, ou a versão sem endosso. Outra desvantagem, do ponto de vista jurídico, é que seus termos são um tanto vagos e não são mapeados diretamente aos termos utilizados no licenciamento de software, havendo um maior esforço de interpretação da licença para demostrar que, de fato, a maioria dos direitos em geral restritos ao autor estão sendo transferidos para o licenciado. Por outro lado, temos como vantagens a simplicidade da licença e sua ampla adoção, que facilitam seu entendimento pelo público geral.

Portanto, o exposto acima faz da licença BSD uma excelente escolha para projetos em que não há uma preocupação a respeito de quais serão os termos que outras pessoas usarão ao redistribuir o trabalho original ou derivado.

5.1.2 A Licença MIT/X11

A Licença MIT (www.opensource.org/licenses/mit-license.php), criada pelo *Massachusetts Institute of Technology*, é também conhecida como Licença X11 ou X, por ter sido redigida originalmente para o *X Window System*, desenvolvido no MIT em 1987.

Essa também é uma licença permissiva e é considerada equivalente à BSD Simplificada sem a cláusula de endosso. Porém, seu texto é bem mais explícito ao tratar dos direitos que estão sendo transferidos, afirmando que qualquer pessoa que obtém uma cópia do software e seus arquivos de documentação associados pode lidar com eles sem restrição,

incluindo sem limitação os direitos a usar, copiar, modificar, mesclar, publicar, distribuir, sublicenciar e/ou vender cópias do software. As condições impostas para tanto são apenas manter o aviso de *copyright* e uma cópia da licença em todas as cópias ou porções substanciais do software.

Para finalizar, a licença contém uma cláusula sobre a ausência de garantias e responsabilidades bastante similar à da licença BSD, protegendo os detentores do direito autoral de qualquer processo judicial relacionado ao software. Nesta licença ainda é excluída explicitamente a responsabilidade de não infração, que pode ocorrer, por exemplo, quando alguém usa a propriedade intelectual de outra pessoa sem a devida autorização. Isso cobre casos em que o autor do software, acidentalmente ou não, tenha utilizado algum material protegido sob direitos autorais ou uma ideia patenteada sem obter uma licença para tanto, o que pode gerar um processo contra o autor, os distribuidores do software e até seus usuários. Porém, assim como explicado anteriormente, a ausência de responsabilidades tem sua validade limitada pelas leis vigentes.

Vantagens e Desvantagens

Essa licença é a recomendada pela *Free Software Foundation* quando se busca uma licença permissiva, pois é bastante conhecida e, ao contrário da BSD, não possui múltiplas versões com cláusulas que podem gerar dificuldades adicionais, tais como a cláusula de propaganda da BSD que pode gerar incompatibilidades com outras licenças. Outra vantagem da MIT em relação à BSD é a maior clareza dos seus termos ao declarar explicitamente que é permitido, por exemplo, sublicenciar ou vender cópias do software, que são direitos apenas implícitos na BSD. A questão do sublicenciamento é bastante importante quando o software será usado como parte de um trabalho coletivo ou derivado que será distribuído sob outra licença, o que é bastante comum nas práticas de software livre. Se não houver o direito de sublicenciamento, então apenas o detentor do direito autoral pode conceder a licença. Desta forma, o usuário de um trabalho derivado precisa obter a licença tanto do autor desse trabalho como também dos detentores dos direitos de cada componente que faz parte dele, sendo necessário identificar todos esses componentes e pessoas envolvidas, aumentando a dificuldade, os riscos e a complexidade jurídicica. Por outro lado, quando é permitido o sublicenciamento, a pessoa que está emitindo a licença, que valerá para todos os componentes de seu software, fica responsável por analisar os termos de cada um dos componentes e certificar-se da compatibilidade entre as licenças. Assim, ao chegar no usuário final, o potencial de problemas é restrito a uma única licença.

5.1.3 A Licença Apache

A Licença Apache está atualmente na versão 2.0 (www.apache.org/licenses/LICENSE-2.0.html) e é usada por um dos projetos mais conhecidos de software livre, o servidor web Apache, assim como pela maior parte dos outros projetos pertencentes à Fundação Apache, além de projetos independentes que optaram por usar essa licença.

A Apache também é uma licença permissiva e, na versão 1.1, seu texto era bastante similar ao da BSD, porém dando ênfase à proteção da marca Apache. Havia uma cláusula similar à cláusula de propaganda da BSD, obrigando que a documentação ou o software, quando redistribuídos, incluíssem a frase "este produto inclui software desenvolvido pela Apache Software Foundation (http://www.apache.org/)" e duas cláusulas proibindo o uso do nome Apache sem permissão escrita prévia, tanto para endossar trabalhos derivados, como na BSD, como também o uso como parte do nome do produto. Em 2004, a licença foi totalmente reescrita e seu texto ficou bem mais longo e complexo, detalhando melhor os direitos concedidos, conforme será visto a seguir.

A primeira parte da licença contém as definições de palavras-chave que serão utilizadas no decorrer da licença, tais como "Entidade Legal", "Você" e "Fonte". A definição de fonte é mais abrangente do que a convencional, englobando não apenas o código fonte do software, como também fonte da documentação e arquivos de configuração. Analogamente, na definição de "Objeto" é incluída qualquer coisa resultante da transformação ou tradução mecânica da fonte, tais como código compilado, documentação e conversões para outros tipos de mídia. Essa licença também define o que é um "Trabalho Derivado", excluindo explicitamente trabalhos que se mantém separados do trabalho sendo licenciado ou que meramente são ligados à sua interface. A próxima definição é a de "Contribuição", que é uma modificação do trabalho original que será incluída no trabalho em futuras versões. Conforme será visto adiante, a contribuição também pode ser licenciada nos termos da licença Apache.

As próximas duas cláusulas se referem à concessão de direitos autorais e patentes. Quanto aos direitos autorais, é dada uma licença irrevogável para reproduzir, preparar trabalhos derivados, mostrar publicamente, sublicenciar e distribuir o trabalho e seus derivados, na forma de fonte ou objeto, sujeitos aos termos e condições da licença. Essa cláusula enumera os direitos protegidos pela lei que estão sendo concedidos à pessoa que obtém a licença, adaptando-se bem às leis americanas, porém, no caso brasileiro, em que a legislação de direitos autorais é muito mais restritiva, acaba deixando de lado alguns direitos fundamentais, como por exemplo o do uso do software (vide Seção 4.2). Nesse sentido, a versão anterior da licença era mais apropriada, já que definia os direitos concedidos de forma mais abrangente e incluia em seu texto a permissão de uso.

Em seguida, são colocadas algumas condições para a redistribuição do trabalho e seus derivados:

- incluir uma cópia da licença;
- incluir avisos em todos os arquivos modificados informando sobre a alteração;
- manter na fonte de trabalhos derivados todos os avisos de direitos autorais, patentes e marcas registradas que são pertinentes;
- se o trabalho incluir um arquivo texto chamado "NOTICE", então qualquer trabalho derivado distribuído deve incluir os avisos pertinentes contidos nesse arquivo da forma como está detalhado na licença.

A cláusula sobre redistribuição termina informando explicitamente que é permitido licenciar sob outros termos qualquer modificação ou trabalho derivado, desde que o uso, reprodução e distribuição do trabalho que foi obtido pela licença Apache sejam respeitados.

Já no caso das contribuições enviadas, que são tratadas na cláusula seguinte, fica determinado que é usada a princípio a licença Apache, sem termos ou condições adicionais, a não ser que tenha sido realizado algum outro tipo de acordo de licenciamento entre as partes.

As quatro últimas cláusulas tratam das marcas registradas e responsabilidades legais. A licença limita-se a permitir o uso da marca apenas para uso razoável, para descrever a origem do trabalho e reproduzir o arquivo NOTICE. Dessa forma, fica implícito que não é permitido associar os nomes relacionados ao trabalho original a trabalhos derivados além do necessário para indicar a origem do trabalho. A cláusula que informa sobre a falta de garantias e responsabilidade é bastante similar àquela vista anteriormente na licença MIT, assim como a cláusula sobre limitações da responsabilidade. A principal diferença está na proteção das contribuições e seus responsáveis, já que esses também são tratados no resto da licença. Na última cláusula, é tratada a questão de outras entidades aceitarem prover garantias ou serem responsabilizadas pelo trabalho, o que é permitido desde que fique claro que a entidade está agindo por conta própria e ficará totalmente responsável pelas ações judiciais decorrentes, de forma a não interferir nas proteções judiciais que recaem sobre os contribuidores de acordo com as cláusulas anteriores da licença.

Após apresentar os termos levantados, a licença inclui instruções sobre como aplicá-la a um trabalho. Como o texto da licença é bastante extenso, tornando inconveniente apresentá-lo por completo em cada arquivo do código fonte do projeto, a licença dá como diretriz que seja usado o seguinte texto:

```
Copyright [ano] [nome do detentor dos direitos]

Licensed under the Apache License, Version 2.0 (the "License"); you may
not use this file except in compliance with the License. You may obtain
a copy of the License at http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an ``AS IS'' BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

Vantagens e Desvantagens

A principal vantagem da licença Apache é seus termos estarem definidos de forma mais precisa, deixando menos margem a interpretações conflitantes com os interesses dos envolvidos. Em particular, o fato da licença Apache deixar explícito na cláusula 4, so-

bre redistribuição, que é permitido o uso de outra licença, é visto como uma vantagem sobre as licenças BSD ou MIT quanto à clareza de sua característica permissiva. Outra vantagem dessa licença é o tratamento de contribuições, tornando mais transparente a incorporação das mesmas em um projeto quando é feita a opção por manter a mesma forma de licenciamento. Para formalizar o processo de contribuições, foi criado o *Apache Contributor License Agreement*, que transmite à *Apache Software Foundation* todos os direitos de propriedade intelectual necessários para que a fundação possa licenciar a contribuição, o que é importante caso ocorra a decisão de uma alteração de licença do projeto. O fato do texto da licença ser mais extenso do que os vistos anteriormente é visto como uma desvantagem por algumas pessoas, porém o ganho em precisão compensa esse fato. O principal problema da Apache é o fato de sua compatibilidade com a GPL 2.0 ser discutível, dado que ela impõe algumas restrições que não estão no texto dessa versão da GPL, conforme será visto no próximo capítulo.

5.2 Recíprocas Totais

As licenças recíprocas totais determinam que qualquer trabalho derivado precisa ser distribuído sob os mesmos termos da licença original. Isso também é chamado de *copyleft*, um termo criado pela *Free Software Foundation* (www.gnu.org/copyleft). A ideia do *copyleft* é dar permissão a todos para executar, copiar, modificar e distribuir versões modificadas do programa, mas impedir que sejam adicionadas restrições a essas versões redistribuídas. Tal ideia visa fortalecer o software livre como um todo, não permitindo que melhorias do software sejam retiradas do alcance da comunidade. O resultado esperado é que a quantidade de software livre aumente cada vez mais, beneficiando todos os envolvidos na cadeia produtiva do software livre. Além disso, a reciprocidade contribui para manter a compatibilidade entre diversas versões de um determinado sistema, dado que quando novas funcionalidades são feitas de forma fechada, fica mais difícil replicá-las nas diferentes versões. Por outro lado, tal abordagem também sofre críticas de dentro da comunidade, pois o software licenciado nesse modelo acaba ficando de certa forma isolado dos demais devido a incompatibilidades nas licenças. Na prática, software licenciado sob o modelo permissivo, em geral, pode ser incorporado em software licenciado como recíproco, já que licenças permissivas permitem a redistribuição sob outros termos, inclusive os de licenças recíprocas. Porém, o inverso não é verdadeiro e, assim, software sob licenças recíprocas não pode ser utilizado em vários projetos de software livre que usam alguma outra licença.

A licença que deu origem à ideia de *copyleft* foi a *General Public License*, ou GPL (www.gnu.org/licenses/gpl.html), da *Free Software Foundation*. Nesta seção, iremos estudar algumas de suas versões: a GPL 2.0, GPLv3 e *Affero General Public License*, ou AGPL.

5.2.1 GPL 2.0

A licença GPL (www.gnu.org/licenses/old-licenses/gpl-1.0.html) foi escrita em 1989 pela *Free Software Foundation*. Dois anos depois, em junho de 1991, foram feitas pequenas modificações na licença, gerando a versão 2.0 (www.gnu.org/licenses/old-licenses/gpl-2.0.html). Essa versão manteve-se até 2007, quando saiu a GPLv3, que será vista na próxima seção. Devido ao grande número de projetos licenciados sob a versão 2.0, incluiremos também nesta dissertação uma descrição detalhada dessa versão da licença.

A GPL 2.0 inclui um preâmbulo que explica os princípios que baseiam a licença e seus principais objetivos. Apesar desse preâmbulo ser bastante citado nas discussões, seu valor jurídico é menor que o restante da licença, pois, por não fazer fazer parte dos termos e condições, suas palavras não precisam ser obedecidas por quem obtém a licença do software. Seu objetivo é apenas melhorar o entendimento da GPL em seu contexto [Ros05], explicando o que é software livre e a importância do *copyleft*. Porém, ele pode ser utilizado para esclarecer a vontade do licenciante no caso de uma disputa ir a tribunal.

A licença GPL pode ser copiada, distribuída e aplicada a qualquer software cujo detentor dos direitos autorais assim desejar. Porém, diferentemente de outras licenças, como a BSD, o texto da GPL não pode ser alterado sem autorização, ou seja, não é permitido que seja feita uma licença derivada dela. No final da licença, há uma explicação sobre como aplicá-la a um trabalho. A *Free Software Foundation* recomenda que o autor que usa a GPL permita que seu trabalho esteja licenciado sob a versão mais recente da licença ou qualquer versão posterior, de forma que quando surgir uma nova versão o usuário da licença possa escolher qual das versões irá utilizar. Dessa forma, evita-se incompatibilidade entre programas mais antigos e mais novos que optaram por utilizar a GPL. Porém, muitas pessoas preferem ter maior controle sobre quais são os termos em que seu software está licenciado e, assim, não deixam aberto para qualquer versão posterior criada pela *Free Software Foundation*. Um exemplo importante dessa escolha é o núcleo do Linux, que será estudado no Capítulo 8.

A Seção 0 da GPL define ao que ela se aplica, que seria qualquer programa ou trabalho que contém uma nota afirmando que está sendo licenciado pelos termos da GPL. No resto da licença, esse objeto a ser licenciado é chamado simplesmente de *Programa*. A Seção 0 também afirma que a licença se aplica a qualquer trabalho derivado segundo a lei de direitos autorais, porém, ao mesmo tempo, define trabalho derivado como "um trabalho contendo o Programa ou parte dele, literalmente ou com modificações e/ou traduzido para outra língua" [Fre91]. Essa definição dada a trabalho derivado difere tanto da legislação americana quanto da brasileira. Segundo a definição dada pela GPL, mesmo trabalhos coletivos são considerados como trabalhos derivados (vide Seção 4.2). Tal definição é de importância fundamental na aplicação da GPL e motivo de muita controvérsia quanto à necessidade de reciprocidade em diferentes usos do software, como será visto no Capítulo 6. Além disso, a Seção 0 afirma que o ato de executar o Programa não é restrito de nenhuma forma e que a saída do Programa só está coberta pela licença se seu conteúdo

constituir de alguma forma um trabalho baseado no Programa, o que só aconteceria em casos muito específicos como, por exemplo, um programa que gera outro programa que é uma versão modificada de si mesmo.

A próxima seção trata da cópia e distribuição do código fonte do Programa, que pode ser realizada desde que sejam mantidos os avisos sobre o *copyright*, a ausência de garantias e a licença. Nesta seção também é explicado que é permitido exigir pagamento pelo ato de transferir uma cópia ou por garantias adicionais que a pessoa decida oferecer, o que permite o uso do software em um modelo de negócio comercial.

A Seção 2 da GPL é uma das mais importantes da licença, pois é onde estão definidas as regras relacionadas ao conceito de *copyleft*, tratando das modificações do programa e sua distribuição. As condições impostas, além daquelas que foram estipuladas para a distribuição segundo o parágrafo anterior, são as seguintes:

1. "Os arquivos modificados precisam conter avisos proeminentes afirmando que os arquivos foram modificados e a data da modificação", de forma a proteger a reputação do autor do trabalho original, que não fica sujeito a problemas causados pela versão modificada.

2. "Qualquer trabalho distribuído ou publicado que contém totalmente ou em parte o programa ou que é derivado do mesmo ou parte dele precisa ser licenciado como um todo sem nenhuma cobrança sob os termos desta licença", o que significa que qualquer trabalho derivado está sujeito às mesmas restrições da GPL. Ou seja, não é possível que um trabalho derivado de um programa GPL seja licenciado de modo fechado, permissivo, ou qualquer outra licença, a não ser que haja a opção de utilizar outra licença para esse programa.

3. "Se o programa modificado normalmente lê comandos de forma interativa quando é executado, é necessário que, quando ele começar a ser executado normalmente, ele imprima ou mostre um aviso incluindo as informações sobre *copyright* e ausência de garantias (ou que uma garantia é provida pela pessoa) e que os usuários podem redistribuir o programa sob essas condições e informando o usuário como ver uma cópia dessa licença". É feita uma exceção: tal regra não precisa ser seguida no caso do programa original ser interativo mas não mostrar tal aviso. O objetivo deste item é garantir que as pessoas sejam informadas sobre seus direitos, de forma que possam utilizá-los da forma como lhes for mais interessante.

Em seguida, a Seção 2 define, com maiores detalhes, as condições em que são aplicadas as regras acima. Porém, as explicações dadas ainda estão longe de esclarecer de fato o impacto da reciprocidade em todos os casos de combinação de um programa GPL com o de outra licença. É explicado que as condições acima aplicam-se ao trabalho modificado como um todo. Os termos da GPL não precisam ser aplicados a seções que possam ser consideradas independentes e não derivadas do programa, desde que tais seções sejam distribuídas como trabalhos separados. Por outro lado, quando essas mesmas seções são

distribuídas como parte de um todo que é um trabalho baseado no programa, então a distribuição desse todo precisa estar nos termos da GPL, cujas permissões se estendem para cada uma de suas partes independentemente de quem a escreveu. Isso implica que, para que o trabalho possa ser distribuído, cada uma de suas partes precisa no mínimo ter uma licença que seja compatível com os termos da GPL, já que no trabalho como um todo tais condições serão aplicadas.

A Seção 10 da GPL, que será vista adiante, provê algumas instruções sobre o que fazer quando as licenças não são compatíveis. Para evitar tal problema, alguns projetos optam por distribuir as partes necessárias para seu funcionamento separadamente, instruindo os usuários sobre como obter as partes que não estão sob seu controle e que poderiam gerar problemas de licença. É explicado que a intenção da Seção 2 não é reivindicar direitos de trabalhos criados inteiramente por outra pessoa, mas sim exercer o direito de controlar a distribuição de trabalhos derivados ou coletivos baseados no Programa. A seção finaliza afirmando que a mera agregação do Programa com outro trabalho que não seja baseado no Programa em um volume de armazenamento ou midia de distribuição não faz com que o outro trabalho caia no escopo desta licença [Fre91]. Isso evita interpretações que exageram nas consequências da licença, limitando seu alcance a trabalhos que realmente são de alguma forma derivados do programa licenciado sob a GPL.

Na próxima seção, é determinado que para distribuir o Programa é necessário que ele esteja acompanhado do código fonte. Como alternativa a distribuir o código junto com o Programa, é permitido que seja feita uma oferta por escrito, com validade mínima de três anos, de que o código seja enviado mediante uma cobrança não superior ao custo de fazer tal distribuição física. Uma pessoa que recebeu tal oferta pode reutilizá-la em sua própria redistribuição do código, desde que seja para fins não-comerciais. Em geral, a não ser que o código seja muito grande, quem distribui programas GPL já disponibiliza o código junto com o binário, de forma a evitar trabalho e custos adicionais tanto para quem está distribuindo como para quem está recebendo. Essa seção define, ainda, que o código fonte inclui todos os módulos que ele contém, arquivos de definição de interface associados e *scripts* usados para controlar a compilação e instalação do executável. Porém, não é necessário incluir partes que normalmente são distribuídas junto com o sistema operacional em que o executável será usado, como por exemplo bibliotecas que já são parte da linguagem de programação utilizada. Essa seção finaliza explicando que se a distribuição do executável é realizada oferecendo acesso para que ele seja copiado de um determinado lugar, então disponibilizar o código fonte no mesmo lugar conta como distribuição do código fonte.

As Seções 4 a 6 da GPL continuam a explicação sobre o funcionamento da licença. Na seção 4 observa-se que, diferentemente de outras licenças que baseiam-se principalmente nas leis de contrato, na GPL há um foco também nas leis de direitos autorais. É ressaltado o fato de que somente a licença permite que uma pessoa modifique ou distribua o Programa ou um derivado dele, já que tais atos são protegidos pelas leis de direito autoral. Se a pessoa não seguir os termos da licença, ela perde esses direitos. Porém,

isso não interfere nas pessoas que receberam os direitos a partir daquela que infringiu a licença. Essas pessoas continuam usufruindo dos diretos desde que elas próprias não cometam nenhuma infração. Isso acontece porque, conforme explicado na Seção 6, quando o Programa é redistribuído, o recipiente automaticamente recebe uma licença do detentor original dos direitos, e não do seu intermediário. Essa relação criada diretamente entre quem emite a licença e quem a recebe permite também que o detentor dos direitos entre com um processo contra qualquer infrator da licença. A Seção 6 também afirma que "você não poderá impor aos recebedores qualquer outra restrição ao exercício dos direitos então adquiridos". Na prática, essa limitação causa a incompatibilidade de outras licenças com a GPL. Devido a isso e às regras impostas na Seção 2, software distribuído sob licenças que têm alguma restrição diferente não pode ser combinado com software GPL.

As Seções 7 e 8 da GPL continuam afirmando a necessidade de seguir os termos da licença, mesmo na presença de outros fatores geradores de outras obrigações, como, por exemplo, decisões judiciais e leis locais. Segundo o que está escrito, caso não seja possível cumprir tais obrigações e ao mesmo tempo seguir os termos da licença, então não é permitido que o Programa seja distribuído. Essa parte da licença é de particular importância em países onde há patentes de software, pois no caso de uma decisão judicial tentar incluir restrições relacionadas a um software, ele não pode mais ser distribuído sob a GPL. Dessa forma, o modelo de software livre proposto pela GPL é garantido mesmo na presença de decisões judiciais que iriam contra seus princípios. É permitido ao detentor dos direitos autorais limitar a distribuição do seu Programa onde as leis locais apresentam conflitos com a GPL. Porém, na prática, existe a possibilidade do juiz determinar que o programa pode ser distribuído ao mesmo tempo em que invalida algum termo da licença.

Para finalizar, as duas seções seguintes da GPL provêem mais duas explicações sobre a licença. A Seção 9 trata da política de versões das licenças publicadas pela *Free Software Foundation* e as possibilidades de escolha da versão da licença conforme o modo como a licença é especificada no Programa. Já a Seção 10 recomenda que, se uma pessoa quer incorporar o Programa em algum outro software livre cujas condições de distribuição sejam diferentes, então ela deve entrar em contato com o autor para tentar conseguir uma permissão específica. A licença termina de forma similar às licenças vistas anteriormente, com duas seções declarando a ausência de garantias. Após os termos e condições, a licença inclui, ainda, alguns parágrafos sobre como aplicá-la ao programa a ser licenciado.

A *Free Software Foundation* informa que bibliotecas que poderão ser incorporadas em software fechado podem utilizar a licença LGPL, que será vista adiante. Segundo ela, a GPL não permite que um programa coberto por essa licença seja ligado a software licenciado sob outros termos, conforme explicado na Seção 2. Tal interpretação é causa de controvérsia entre os estudiosos de licenças de software livre. Lawrence Rosen, em seu livro *Open Source Licensing* [Ros05], uma das principais fontes sobre o assunto, argumenta que o uso de uma biblioteca cria um trabalho coletivo e não um trabalho derivado. Portanto, não seria necessário licenciar o software como LGPL quando distribuído separadamente. Como essa interpretação é contrária à da *Free Software Foundation*, autora da GPL,

Rosen afirma ainda que o que importa é o entendimento entre o detentor dos direitos e a pessoa que recebe a licença. Ele cita como exemplo o caso do núcleo do Linux, em que seu autor, Linus Torvalds, apesar de usar a GPL, adotou a política de que software que é apenas combinado com o Linux não está sujeito aos termos da GPL, independentemente de como esse software é ligado ou distribuído. Dessa forma é possível a existência de módulos fechados no Linux. Porém, considerando a legislação brasileira, essa interpretação de trabalho coletivo não é cabível.

Vantagens e Desvantagens

A GPL é bastante conhecida, sendo a licença mais utilizada em projetos de software livre. Porém, é considerada uma licença de alta complexidade. Apesar da intenção da licença estar clara em seu preâmbulo, há vários detalhes presentes em seus termos que dificultam sua interpretação em casos específicos.

A GPL é recomendada para projetos que buscam seu crescimento através de contribuições de terceiros, dado que melhorias feitas ao software devem manter-se livres para poderem ser distribuídas. A GPL também é usada frequentemente em um modelo comercial de licenciamento duplo. Neste caso, a empresa provê o software sob a licença GPL, obtendo os benefícios relacionados ao software livre, mas ao mesmo tempo disponibiliza o software sob alguma outra licença que não imponha as restrições presentes na GPL. Dessa forma, empresas que têm interesse em usar o software de forma fechada podem obter uma licença alternativa, normalmente pagando um determinado valor para a empresa detentora dos direitos sobre o software.

Por fim, antes de adotar a GPL, é muito importante verificar sua compatibilidade com licenças de outros programas que serão utilizados no projeto, de forma a evitar ter que reescrever partes do software que já estariam disponíveis sob alguma outra licença.

5.2.2 GPLv3

A mais nova versão da GPL (www.gnu.org/licenses/gpl-3.0-standalone.html), lançada em 29 de junho de 2007, após um longo período de discussão e revisão pública, foi criada para evitar algumas situações consideradas indesejáveis pela *Free Software Foundation*. Além disso, algumas partes foram reescritas de forma a adaptar a licença a novas formas de compartilhamento de programa e a deixá-la mais adequada para legislações em que os termos originais poderiam ser interpretados de forma diferente da esperada pela *Free Software Foundation*. Também foram realizadas alterações para facilitar a compatibilidade com outras licenças, em particular a Apache 2.0 [Laf07]. A seguir serão discutidas as principais mudanças.

Um dos principais motivos para mudar para a GPLv3, segundo seus criadores, é evitar o fenômeno conhecido como *tivoização* (em referência ao aparelho TiVo, que funciona como um gravador digital de vídeos). O TiVo inclui software derivado do Linux, licenciado sob a GPL 2.0. O código está disponível e pode ser modificado, porém, tais modificações

não podem ser utilizadas no aparelho TiVo, pois ele faz uma checagem da assinatura digital do software e executa apenas as versões permitidas pelo fabricante. Essa questão de assinaturas digitais foi bastante controversa durante a elaboração da GPLv3, pois ao mesmo tempo em que as assinaturas restringem a liberdade dos usuários, defendida pela *Free Software Foundation*, elas são uma ferramenta importante para implementar segurança em alguns sistemas. A estratégia para impedir a *tivoização* é exigir que o fabricante provenha toda informação necessária para instalar versões modificadas do software no aparelho. Essa informação pode ir desde instruções básicas até chaves de autorização que possam ser necessárias. Porém, tal exigência é limitada a aparelhos considerados "produtos de usuário". Assim, alguns equipamentos de uso específico, como por exemplo máquinas de votação, estariam isentos da necessidade de reduzir seu nível de segurança para se adequar a licença. A definição de "produtos de usuário" está presente na Seção 6 da licença, sobre distribuição na forma binária:

```
A "User Product" is either (1) a "consumer product", which means any
tangible personal property which is normally used for personal, family,
or household purposes, or (2) anything designed or sold for
incorporation into a dwelling. In determining whether a product is a
consumer product, doubtful cases shall be resolved in favor of
coverage. For a particular product received by a particular user,
"normally used" refers to a typical or common use of that class of
product, regardless of the status of the particular user or of the way
in which the particular user actually uses, or expects or is expected
to use, the product. A product is a consumer product regardless of
whether the product has substantial commercial, industrial or
non-consumer uses, unless such uses represent the only significant
mode of use of the product.
```

Outra mudança na GPLv3 é relacionada a mecanismos de DRM, ou *Digital Rights Management* (www.defectivebydesign.org). É sabido que a *Free Software Foundation* é contra o uso de DRM, porém, optou por não impedir que software livre fosse utilizado para implementá-lo, já que isso estaria limitando a liberdade dos usuários. Como alternativa, decidiram atacar o tratado de *copyright* da *World Intellectual Property Organization* (WIPO), adotado em 20 de dezembro de 1996. Nesse tratado, o artigo 11, sobre obrigações a respeito de medidas tecnológicas, afirma que os países devem adotar medidas legais para reprimir tentativas de burlar sistemas tecnológicos de proteção usados por autores para restringir atos que não são autorizados, em conexão com o exercício de seus direitos de acordo com esse tratado ou a Convenção de Berna. A solução presente na GPLv3 é afirmar que qualquer trabalho sob a GPL não pode ser considerado uma "medida tecnológica efetiva" contra quebra de DRM. Ou seja, é permitido incluir sistemas de DRM no software tanto quanto é permitido quebrá-lo. Tal cláusula está na seção três:

```
3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological
measure under any applicable law fulfilling obligations under article 11
of the WIPO copyright treaty adopted on 20 December 1996, or similar
laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid
circumvention of technological measures to the extent such circumvention
is effected by exercising rights under this License with respect to the
covered work, and you disclaim any intention to limit operation or
modification of the work as a means of enforcing, against the work's
users, your or third parties' legal rights to forbid circumvention of
technological measures.
```

Outro ponto tratado em maior profundidade na nova versão da GPL é a questão das patentes. Uma das motivações para as mudanças que foram implementadas na licença foi um acordo entre a Microsoft e a Novell, decorrente de uma alegação da Microsoft de que a Novell estaria infringindo suas patentes na distribuição SUSE Linux. Segundo o acordo, a Microsoft não processaria usuários por infração de patentes desde que o software fosse obtido de alguém que estivesse pagando à Microsoft para obter tais direitos. A GPLv3 é bem específica quanto ao caso e, na Seção 11, sobre patentes, afirma que uma organização não poderá distribuir trabalhos cobertos por essa licença caso faça parte de um desses acordos discriminatórios, conforme detalhado a seguir.

```
A patent license is "discriminatory" if it does not include within the
scope of its coverage, prohibits the exercise of, or is conditioned on
the non-exercise of one or more of the rights that are specifically
granted under this License. You may not convey a covered work if
you are a party to an arrangement with a third party that is in the
business of distributing software, under which you make payment to
the third party based on the extent of your activity of conveying the
work, and under which the third party grants, to any of the parties
who would receive the covered work from you, a discriminatory
patent license (a) in connection with copies of the covered work
conveyed by you (or copies made from those copies), or
(b) primarily for and in connection with specific products or
compilations that contain the covered work, unless you entered
into that arrangement, or that patent license was granted, prior
to 28 March 2007.
```

Há ainda outras definições sobre patentes nessa seção, porém, elas não serão detalhadas aqui, já que não são relevantes dentro da legislação brasileira, conforme visto no Capítulo 4. Mas essa questão das patentes é uma das principais causas de receio por parte das empresas que detêm propriedade intelectual nessa forma. Como a comunidade de software livre muitas vezes busca o apoio dessas empresas, esse tornou-se um forte

fator para frear a adoção da GPLv3.

Quanto à compatibilidade com outras licenças, a mais importante delas sendo a Apache, foram adicionados alguns termos na Seção 7 da GPL, intitulada "Termos Adicionais", de forma a permitir que software cuja licença possua certas restrições ainda assim possa ser relicenciado sob a GPL, e dessa forma ser distribuído como parte de um trabalho derivado ou coletivo:

```
Notwithstanding any other provision of this License, for material you
add to a covered work, you may (if authorized by the copyright holders
of that material) supplement the terms of this License with terms:

    * a) Disclaiming warranty or limiting liability differently from the
        terms of sections 15 and 16 of this License; or

    * b) Requiring preservation of specified reasonable legal notices
        or author attributions in that material or in the Appropriate
        Legal Notices displayed by works containing it; or

    * c) Prohibiting misrepresentation of the origin of that material,
        or requiring that modified versions of such material be marked
        in reasonable ways as different from the original version; or

    * d) Limiting the use for publicity purposes of names of licensors
        or authors of the material; or

    * e) Declining to grant rights under trademark law for use of some
        trade names, trademarks, or service marks; or

    * f) Requiring indemnification of licensors and authors of that
        material by anyone who conveys the material (or modified
        versions of it) with contractual assumptions of liability to the
        recipient, for any liability that these contractual assumptions
        directly impose on those licensors  and authors.
```

Para a distribuição de código binário, foram adicionadas novas possibilidades de disponibilização do código fonte correspondente, a mais importante delas tratando do compartilhamento *peer-to-peer*. Nesse caso, basta que seja informado onde o código fonte está sendo oferecido ao público. Além disso, segundo a Seção 9, a mera transmissão do programa através de *peer-to-peer* não obriga que o usuário aceite a licença. Ou seja, apenas a pessoa que inicia o processo de distribuição *peer-to-peer* precisa se preocupar em informar sobre o código fonte, as demais estando isentas de qualquer obrigação.

Por fim, vale comentar que há algumas mudanças sutis na forma como a licença foi escrita, visando evitar ambiguidade de seus termos. Por exemplo, a palavra *distribute* foi substituída por *convey*, termo que foi definido no início da licença como "qualquer forma de propagação que permite a outras partes fazer ou receber cópias".

Vantagens e Desvantagens

Considerando que essa é apenas uma atualização da GPL, as vantagens e desvantagens analisadas aqui serão em relação a sua versão anterior, que já foi discutida. A GPLv3 foi lançada para corrigir algumas brechas que foram verificadas no decorrer dos anos com o uso da GPL 2.0, deixando a licença mais alinhada à visão de *copyleft* da *Free Software Foundation*.

Seu texto passou por um longo período de revisão aberta pela comunidade, em que foi possível sugerir e incorporar diversas melhorias. Sendo assim, é um texto bastante sólido e bem escrito, que está consistente com uma ampla gama de legislações, inclusive a brasileira. A revisão da GPL também serviu para alinhá-la às atuais práticas de distribuição de software, incluindo o compartilhamento em redes *peer-to-peer*.

Uma das principais vantagens da nova versão da GPL é sua compatibilidade com um maior número de licenças de software livre, principalmente a Apache, que é uma das licenças permissivas mais usadas pela comunidade. Porém, a GPLv3 também apresenta uma forte desvantagem em relação a compatibilidade: software que está sob a licença GPL 2.0 (sem a cláusula "ou posterior") não pode ser integrado a software GPLv3, pois essas licenças são incompatíveis.

Outra desvantagem é que várias empresas têm um certo receio em adotar a GPLv3, pois ela é uma licença relativamente nova e de alta complexidade. A cláusula que gera maior preocupação nos advogados corporativos é a relacionada às patentes.

Portanto, se o objetivo é garantir as liberdades propostas pelo movimento de software livre, a GPLv3 é mais adequada do que sua anterior. Porém, se uma maior difusão do software é o mais importante, tornando-o compatível com um maior número de licenças e incentivando seu uso em qualquer empresa, então deixar sob a licença "GPL versão 2 ou superior" pode ser uma melhor alternativa. Note que se o projeto envolver componentes cuja licença é compatível apenas com a GPLv3, então é necessário que a licença adotada seja a GPLv3, e não "GPL versão 2 ou superior".

5.2.3 AGPL

A Affero Inc. (www.affero.org) é uma empresa que se define com a missão de "trazer a cultura de patrocínio para a Internet". Ela provê um serviço de hospedagem de páginas pessoais para autores de diversos tipos e integra um sistema de pagamento seguro para que pessoas possam fazer doações. A Affero apóia o desenvolvimento de software livre e, em março de 2002, criou a primeira versão da *Affero General Public License*, ou AGPL (www.affero.org/oagpl.html). A AGPL é uma adaptação da GPL, autorizada pela *Free Software Foundation*, que inclui um termo sobre uso de um software através de uma rede. O parágrafo adicionado à Seção 2, sobre modificação e cópia do programa e sua distribuição, é o seguinte:

```
d) If the Program as you received it is intended to interact  with users
through a computer network and if, in the version you received, any user
interacting with the Program was given the opportunity to request
transmission to that user of the Program's complete source code, you
must not remove that facility from your modified version of the Program
or work based on the Program, and must offer an equivalent opportunity
for all users interacting with your Program through a computer network
to request immediate transmission by HTTP of the complete source
code of your modified version or other derivative work.
```

Esse parágrafo afirma, de forma resumida, que se no programa original os usuários que interagiam com o programa tinham a opção de pedir o código fonte completo, tal opção tem que ser mantida em qualquer versão modificada. Dessa forma, mesmo não havendo a distribuição de um binário, um aplicativo web público sob esta licença precisa se manter aberto para qualquer usuário que interaja com ele.

Em 19 de novembro de 2007, a *Free Software Foundation* lançou a *GNU Affero General Public License*, conhecida como AGPLv3 (www.fsf.org/licensing/licenses/agpl-3.0.html), uma licença diferente daquela escrita pela Affero e compatível com a GPLv3. Para permitir que um programa licenciado sob a antiga licença Affero que utilizasse a cláusula "ou posterior" pudesse ser convertido para essa nova licença, a Affero Inc. criou uma versão intermediária da licença, que determina apenas o seguinte:

```
This is version 2 of the Affero General Public License. It gives each
licensee permission to distribute the Program or a work based on the
Program (as defined in version 1 of the Affero GPL) under the GNU Affero
General Public License, version 3 or any later version.

If the Program was licensed under version 1 of the Affero GPL "or any
later version", no additional obligations are imposed on any author or
copyright holder of the Program as a result of a licensee's choice to
follow this version 2 of the Affero GPL.
```

As diferenças entre a GPLv3 e a AGPLv3 residem no preâmbulo e na seção 13 da licença. Enquanto na GPLv3 a seção 13 informa apenas que trabalhos que usam essas duas licenças podem ser combinados, valendo a AGPLv3 como licença do trabalho resultante, a seção 13 da AGPLv3 afirma ainda que:

```
Notwithstanding any other provision of this License, if you modify the
Program, your modified version must prominently offer all users
interacting with it remotely through a computer network (if your version
supports such interaction) an opportunity to receive the Corresponding
Source of your version by providing access to the Corresponding Source
```

```
from a network server at no charge, through some standard or customary
means of facilitating copying of software. This Corresponding Source
shall include the Corresponding Source for any work covered by version 3
of the GNU General Public License that is incorporated pursuant to the
following paragraph.
```

Essa nova versão da redação da cláusula visa tornar a Affero GPL ainda mais abrangente. Enquanto originalmente era colocado como critério que o programa fosse feito para interagir com usuários através de uma rede, nessa nova versão, qualquer tipo de software que interage através da rede está coberto, como por exemplo servidores de jogos, que interagem com o usuário através de um outro programa intermediário.

Vantagens e Desvantagens

Para a AGPL valem todas as considerações feitas a respeito da GPL. Ela é recomendada para projetos em que há interação via rede e busca-se o *copyleft*. A AGPL é considerada a mais "viral" das licenças, portanto deve ser evitada em projetos em que haja qualquer expectativa de utilização sob outra licença, a não ser que seja adotado um modelo de licenciamento múltiplo.

5.2.4 European Union Public License

A União Européia lançou, em janeiro de 2007, a *European Union Public License 1.0* (EUPL), baseada na *Open Software License* (OSL) versões 2.1 e 3.0.

Os principais motivadores para a criação da EUPL foram:

- fomentar o reconhecimento, uso e distribuição de software livre, em particular entre as administrações públicas européias;
- ter licenças válidas como contrato em 22 línguas oficiais ao invés de usar traduções que têm valor apenas informativo;
- adaptar as licenças considerando as leis européias;
- reduzir problemas de compatiblidade entre licenças *copyleft*.

A licença inicialmente não foi aprovada pelo comitê da *Open Source Initiative*, porém foram sugeridas modificações para uma nova versão da EUPL, a 1.1, que foi aprovada em Março de 2009.

Vantagens e Desvantagens

A licença EUPL é vantajosa para projetos europeus por ter sido adaptada para a legislação da União Européia e por apresentar traduções oficiais para diversos idiomas. A principal desvantagem está no fato de ser uma licença ainda muito nova e pouco conhecida, gerando as dificuldades citadas anteriormente relativas a proliferação de licenças.

5.3 Recíprocas Parciais

Licenças recíprocas parciais, também chamadas de *copyleft fraco*, determinam que modificações do trabalho coberto por elas devem ser disponibilizadas sob a mesma licença. Porém, quando o trabalho é utilizado apenas como um componente de outro projeto, esse projeto não precisa estar sob a mesma licença. Alguns autores, como Simon Phipps [Sun06], utilizam a denominação licença baseada em arquivos para essa categoria, enquanto as recíprocas totais seriam licenças baseadas em projeto.

Considera-se que essas licenças são as que melhor equilibram dois fatores importantes do modelo de software livre: atração de interesse para a comunidade e força e longevidade do código fonte disponível. Ao mesmo tempo que essas licenças permitem que os desenvolvedores utilizem o trabalho para criar software que será licenciado como preferirem, modificações e melhorias feitas ao próprio trabalho são obrigatoriamente disponibilizadas à comunidade [Sun06].

A *Free Software Foundation* recomenda o uso desse tipo de licença apenas em casos específicos. Seu argumento é a necessidade de fortalecer o software livre em detrimento do software fechado. Assim, quando as funcionalidades de uma biblioteca não estão facilmente disponíveis para uso em software fechado, seria melhor mantê-las dessa forma, utilizando uma licença recíproca total. Dessa forma, o software livre teria uma vantagem sobre concorrentes fechados. Porém, se as funcionalidades já estão ao alcance de software fechado, e portanto essa vantagem não está em questão, então uma licença recíproca parcial é recomendada, pois esse modelo ajuda a aumentar o número de usuários da biblioteca [Fre08].

Alguns advogados, como Lawrence Rosen [Ros05], defendem que o uso de bibliotecas que são apenas ligadas a um novo software não caracterizaria um trabalho derivado, mas sim um trabalho coletivo. Ele faz uma analogia a páginas na Web, em que cada uma é um trabalho com direito autoral individual, apesar de muitas vezes estarem presentes ligações de uma para a outra. Segundo ele, esse tipo de relação consiste em um trabalho coletivo. Portanto, nesse cenário, mesmo um software sob uma licença recíproca total poderia ser usado como biblioteca de outro que estaria sob outra licença. Porém, no caso brasileiro, a Lei de Direito Autoral é mais específica e impõe maiores limitações (vide Seção 4.2). Dessa forma, o uso de licenças recíprocas parciais faz-se necessário em casos nos quais o autor quer garantir que o desenvolvimento da biblioteca seja feito no modelo de software livre mas ao mesmo tempo quer permitir seu uso em projetos que utilizam outras licenças.

5.3.1 A Licença LGPL

A *GNU Lesser General Public License*, ou LGPL (www.fsf.org/licensing/licenses/lgpl.html), originalmente denominada *GNU Library General Public License*, foi escrita em 1991 pela *Free Software Foundation*. Assim como a GPL e a AGPL, passou por grandes modificações no final de 2007 para adequar-se à versão 3 das licenças.

Em versões anteriores, a LGPL era uma cópia da GPL com algumas modificações relativas a bibliotecas, definidas na licença como "uma coleção de funções de software e/ou dados preparada para ser convenientemente ligada com programas aplicativos (que usam algumas dessas funções e dados) para formar executáveis." Veremos a seguir os termos da versão 2.1 dessa licença. Nessa parte do capítulo, adotaremos a versão 2.0 da GPL quando esta for mencionada.

A primeira limitação presente na LGPL que não constava na GPL é que trabalhos modificados da biblioteca precisam ser também bibliotecas, segundo o artigo *a* da seção 2. Já o artigo *d* da mesma seção busca maximizar a utilidade da biblioteca, desvinculando-a de aplicações específicas. Ele determina que:

```
If a facility in the modified Library refers to a function or a table
of data to be supplied by an application program that uses the
facility, other than as an argument passed when the facility is
invoked, then you must make a good faith effort to ensure that,
in the event an application does not supply such function or
table, the facility still operates, and performs whatever part of
its purpose remains meaningful.
```

Para permitir a distribuição de bibliotecas LGPL junto com software GPL, há também uma cláusula na LGPL que afirma que pode-se optar por aplicar os termos da GPL ao invés dessa licença (a LGPL) para uma determinada cópia da biblioteca. Porém, é dito também que tal mudança é irreversível para aquela cópia, e assim a GPL é aplicada a todas as cópias subsequentes e trabalhos derivados feitos a partir dela. Além disso, caso seja feito um trabalho derivado da biblioteca que não resulta em uma biblioteca, relicenciá-lo para GPL é a única alternativa.

A Seção 5 da LGPL afirma que trabalhos que apenas usam a biblioteca, considerados isoladamente, não estão sujeitos aos termos da licença. Porém, também descreve em detalhes vários casos de uso da biblioteca em que é necessário estar atento às condições da licença. A Seção 5 está copiada na íntegra a seguir e será discutida em maiores detalhes no Capítulo 6, sobre compatibilidade.

```
5. A program that contains no derivative of any portion of the Library,
but is designed to work with the Library by being compiled or linked
with it, is called a "work that uses the Library". Such a work, in
isolation, is not a derivative work of the Library, and therefore falls
outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates
an executable that is a derivative of the Library (because it contains
portions of the Library), rather than a "work that uses the library".
The executable is therefore covered by this License. Section 6 states
terms for distribution of such executables.
```

```
When a "work that uses the Library" uses material from a header file
that is part of the Library, the object code for the work may be a
derivative work of the Library even though the source code is not.
Whether this is true is especially significant if the work can be
linked without the Library, or if the work is itself a library. The
threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure
layouts and accessors, and small macros and small inline functions
(ten lines or less in length), then the use of the object file is
unrestricted, regardless of whether it is legally a derivative work.
(Executables containing this object code plus portions of the Library
will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may
distribute the object code for the work under the terms of Section 6.
Any executables containing that work also fall under Section 6,
whether or not they are linked directly with the Library itself.
```

Apesar dessa tentativa de especificar o que seria um “trabalho que usa a biblioteca”, em que não há restrições quanto ao licenciamento, em oposição a um “trabalho baseado na biblioteca”, que estaria sujeito aos termos da LGPL, há muitos casos em que tal diferenciação não está clara. Além dos casos omissos, é possível mais de uma interpretação para alguns casos abordados nessa seção.

Há ainda a Seção 6, que faz uma exceção às anteriores e provê termos adicionais para trabalhos que incluem partes da biblioteca na versão que será distribuída. Essa seção é em parte equivalente à Seção 3 da GPL, e sua leitura é recomendada. Segundo os termos dessa seção, se a pessoa que está distribuindo um programa que usa a biblioteca não tem permissão para distribuir algum dos componentes necessários para execução do programa e esse componente não é parte integrante do sistema operacional, então a distribuição do programa na forma executável não é permitida nos termos da LGPL.

Na próxima seção, é descrito o caso de juntar uma biblioteca coberta pela LGPL com alguma outra biblioteca, sob outra licença, para formar uma única biblioteca. As regras impostas para tanto são as seguintes: em primeiro lugar, é necessário que os termos das licenças permitam que as bibliotecas sejam distribuídas como trabalhos separados. Além disso, a parte da biblioteca que está sob LGPL deve estar disponível separadamente sob os termos da LGPL, ou sendo distribuída junto com o conjunto, ou sendo colocado um aviso informando onde obtê-la.

As demais seções da LGPL correspondem às seções 4 a 12 da GPL, apenas com as alterações necessárias para tratar de uma “Biblioteca” ao invés de um “Programa”, conforme definido nas respectivas licenças.

A nova versão da licença, LGPLv3, apresenta-se como um conjunto de termos adicionais à GPLv3, já discutida na Seção 5.2.2. Isso implica que a licença LGPL foi totalmente reescrita em relação à versão 2.1, porém seus princípios foram mantidos. As diferenças en-

tre a LGPL 2.1 e a GPL 2.0 correspondem, em grande parte, do ponto de vista semântico, às permissões adicionais que constituem a LGPLv3.

Há seis cláusulas na LGPLv3. A primeira constitui as "definições adicionais", que explica o que será entendido por "biblioteca", "aplicação", "trabalho combinado", "código fonte correspondente mínimo" e "código da aplicação correspondente".

Em seguida, é explicado como e em que circunstâncias uma pessoa pode distribuir um trabalho que está sob a LGPLv3 sem estar sujeito à seção 3 da GPLv3 (vide termos 1, 2, 3 e 4 da licença, que não serão citados aqui por se tratar de um texto muito extenso, porém cujos detalhes são de suma importância para o uso correto da LGPLv3):

Também são definidos os termos para criar um trabalho constituído por bibliotecas combinadas:

```
5. Combined Libraries.

You may place library facilities that are a work based on the Library
side by side in a single library together with other library facilities
that are not Applications and are not covered by this License, and
convey such a combined library under terms of your choice, if you do
both of the following:

a) Accompany the combined library with a copy of the same work based on
the Library, uncombined with any other library facilities, conveyed
under the terms of this License.

b) Give prominent notice with the combined library that part of it is a
work based on the Library, and explaining where to find the accompanying
uncombined form of the same work.
```

E o último termo da LGPLv3 trata de versões revisadas da licença, informando que a *Free Software Foundation* poderá publicar versões revisadas e/ou novas da LGPL e explicando que se a pessoa recebeu o trabalho licenciado sob uma certa versão da licença "ou qualquer versão posterior", então ela poderá escolher quais das versões da licença ela irá seguir. Se não é especificada uma versão, então a pessoa pode escolher qualquer versão já publicada.

Vantagens e Desvantagens

A LGPL é uma licença de alta complexidade, que requer uma observação bastante atenta dos seus termos para evitar seu descumprimento, que pode acarretar em uma ação judicial. Contextos de uso da biblioteca diferentes em geral requerem ações diferentes por parte da pessoa usando a biblioteca. Apesar de todos os detalhes presentes na licença, há ainda muita margem para interpretação de casos que não estão bem definidos. Além disso, o próprio relicenciamento de trabalhos derivados como LGPL é limitado, às vezes forçando o uso da GPL, como foi visto anteriormente.

Apesar da dificuldade em usar a LGPL, ela é uma licença amplamente adotada, pois combina características permissivas e recíprocas de forma balanceada, trazendo vantagens de ambos os modelos, conforme foi discutido na introdução desta seção.

A escolha entre a versão 2.0 ou a LGPLv3 pode ser tomada com base nos mesmos argumentos apresentados na Seção 5.2.2, sobre a GPLv3

5.3.2 A Licença Mozilla

A *Mozilla Public License*, ou MPL, foi escrita por uma das executivas da Netscape, Mitchell Baker, que tornou-se uma das principais responsáveis pelo projeto Mozilla, atuando como CEO da Fundação Mozilla por um longo período.

A licença Mozilla é considerada bem escrita e serviu como modelo para muitas das licenças de software livre comerciais que a seguiram [Ros05]. Ela une características de licenças recíprocas e de licenças permissivas, e, portanto, também é categorizada como uma licença recíproca parcial. Na licença Mozilla, a delimitação é bastante clara: o código coberto pela licença deve ser redistribuído pelos termos da licença Mozilla, porém esse código também pode ser utilizado em trabalhos ampliados, que podem estar sob outra licença.

A primeira seção da licença, como em um contrato convencional, trata das definições, que são bastante precisas. A lista completa de definições pode ser vista abaixo, porém algumas merecem atenção especial:

- "uso comercial" significa qualquer distribuição ou outra forma de deixar o software disponível, não se limitando ao uso por empresas;
- "contribuidor" recebe uma definição especial nessa licença, diferindo-o tanto do desenvolvedor inicial como também dos usuários comuns que estão usando o projeto;
- "executável" é definido de forma ampla, como qualquer coisa que não é o código fonte;
- "código fonte" é definido em mais detalhes do que encontramos nas licenças vistas anteriormente. São permitidos *patches* e também comprimir o arquivo, desde que o software para descompressão esteja largamente disponível gratuitamente.

```
1. Definitions

1.0.1. "Commercial Use" means distribution or otherwise making the
Covered Code available to a third party.

1.1. "Contributor" means each entity that creates or contributes to the
creation of Modifications.

1.2. "Contributor Version" means the combination of the Original Code,
prior Modifications used by a Contributor, and the Modifications made by
that particular Contributor.
```

1.3. "Covered Code" means the Original Code or Modifications or the combination of the Original Code and Modifications, in each case including portions thereof.

1.4. "Electronic Distribution Mechanism" means a mechanism generally accepted in the software development community for the electronic transfer of data.

1.5. "Executable" means Covered Code in any form other than Source Code.

1.6. "Initial Developer" means the individual or entity identified as the Initial Developer in the Source Code notice required by Exhibit A.

1.7. "Larger Work" means a work which combines Covered Code or portions thereof with code not governed by the terms of this License.

1.8. "License" means this document.

1.8.1. "Licensable" means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently acquired, any and all of the rights conveyed herein.

1.9. "Modifications" means any addition to or deletion from the substance or structure of either the Original Code or any previous Modifications. When Covered Code is released as a series of files, a Modification is:

A. Any addition to or deletion from the contents of a file containing Original Code or previous Modifications.

B. Any new file that contains any part of the Original Code or previous Modifications.

1.10. "Original Code" means Source Code of computer software code which is described in the Source Code notice required by Exhibit A as Original Code, and which, at the time of its release under this License is not already Covered Code governed by this License.

1.10.1. "Patent Claims" means any patent claim(s), now owned or hereafter acquired, including without limitation, method, process, and apparatus claims, in any patent Licensable by grantor.

```
1.11. "Source Code" means the preferred form of the Covered Code for
making modifications to it, including all modules it contains, plus any
associated interface definition files, scripts used to control
compilation and installation of an Executable, or source code
differential comparisons against either the Original Code or another
well known, available Covered Code of the Contributor's choice. The
Source Code can be in a compressed or archival form, provided the
appropriate decompression or de-archiving software is widely available
for no charge.

1.12. "You" (or "Your")  means an individual or a legal entity
exercising rights under, and complying with all of the terms of, this
License or a future version of this License issued under Section 6.1.
For legal entities, "You" includes any entity which controls, is
controlled by, or is under common control with You. For purposes of
this definition, "control" means (a) the power, direct or indirect, to
cause the direction or management of such entity, whether by contract
or otherwise, or (b) ownership of more than fifty percent (50%)
of the outstanding shares or beneficial ownership of such entity.
```

A Seção 2 da MPL trata do licenciamento do código fonte, afirmando que o desenvolvedor inicial está concedendo uma licença não exclusiva, livre de *royalties* e válida em todo mundo. Essa licença libera a propriedade intelectual (exceto patentes ou marcas registradas) para o uso, modificação, reprodução, exibição, performance, sublicenciamento e distribuição do código original (ou porções dele) com ou sem modificações e como parte ou não de um trabalho ampliado. Para o caso das patentes também são concedidas algumas permissões, porém, diferentemente da LGPL, aplicam-se apenas ao código original e não a modificações feitas nele. Nessa seção também constam as permissões concedidas por pessoas que contribuem modificações ao código, que são bastante similares às vistas anteriormente, concedidas pelo desenvolvedor inicial.

A Seção 3 trata das obrigações no ato da distribuição, que são similares às da GPL. As principais exigências são que:

- o código coberto seja distribuído sob os termos da MPL;
- o código fonte esteja disponível;
- as modificações estejam explícitas;
- o aviso legal esteja presente no código fonte;
- na distribuição de formas executáveis, as condições anteriores sejam cumpridas para o código fonte correspondente.

A principal diferença em relação à GPL está na Seção 3.7, referente a trabalhos ampliados, que são criados combinando código coberto com outro código que não está sob a

licença MPL em um único produto. Nesse caso, os requerimentos da MPL recaem apenas ao código coberto, e não ao trabalho como um todo, como seria na GPL.

A Seção 4 da MPL apresenta mais uma diferença em relação à GPL. Na impossibilidade de cumprir algum dos termos da licença por questões judiciais, basta que a situação seja explicada em um arquivo chamado LEGAL e que os demais termos sejam respeitados. Assim, o uso do código não é impedido, como aconteceria com a GPL. Além disso, na Seção 8, é dado um prazo de 30 dias para que seja corrigido qualquer descumprimento dos termos antes que a licença seja terminada.

A Seção 6 afirma que a Netscape tem o direito de alterar a licença a qualquer momento, e que a pessoa que está exercendo os direitos garantidos pela licença pode escolher se seguirá os termos em que o trabalho foi licenciado ou a nova versão. Isso significa que, por outro lado, o desenvolvedor é obrigado a aceitar as modificações de licença como termos válidos para seu projeto que foi licenciado anteriormente. A Netscape transferiu os direitos de alterar a licença para a Fundação Mozilla em 2003, apesar do texto da licença ainda não ter sido modificado para refletir essa alteração. A Seção 6 permite ainda que seja criada uma licença derivada da MPL, evitando assim que a Mozilla tenha controle sobre os termos de licenciamento do projeto, desde que fique claro na versão modificada que ela não está associada à Mozilla ou à Netscape. Além disso, a Seção 13 explica o licenciamento múltiplo, que permite ao usuário escolher qual licença, entre as fornecidas pelo desenvolvedor, ele irá adotar.

As Seções 7, 9, 10, 11 e 12 tratam de garantias, responsabilidade e outros aspectos jurídicos relacionados. Para finalizar, a MPL apresenta o seguinte quadro, para aplicação da licença:

```
EXHIBIT A -Mozilla Public License.

``The contents of this file are subject to the Mozilla Public License
Version 1.1 (the "License"); you may not use this file except in
compliance with the License. You may obtain a copy of the License at
http://www.mozilla.org/MPL/

Software distributed under the License is distributed on an "AS IS"
basis, WITHOUT WARRANTY OF ANY KIND, either express or
implied. See the License for the specific language governing rights
and limitations under the License.

The Original Code is _____.

The Initial Developer of the Original Code is _____.
Portions created by _____ are Copyright (C) _____
_____. All Rights Reserved.

Contributor(s): _____.
```

```
Alternatively, the contents of this file may be used under the terms of
the _____ license (the  "[___] License"), in which case the provisions
of [_______] License are applicable instead of those above.  If you wish
to allow use of your version of this file only under the terms of the
[_____] License and not to allow others to use your version of this file
under the MPL, indicate your decision by deleting  the provisions
above and replace them with the notice and other provisions required
by the [___] License.  If you do not delete the provisions above, a
recipient may use your version of this file under either the MPL or
the [___] License."

[NOTE: The text of this Exhibit A may differ slightly from the text of
the notices in the Source Code files of the Original Code. You should
use the text of this Exhibit A rather than the text found in the Original
Code Source Code for Your Modifications.]
```

Vantagens e Desvantagens

A licença Mozilla encoraja trabalhos ampliados. Até mesmo alguns tipos de projetos que seriam considerados trabalhos derivados pela lei de *copyright* podem usar outra licença. Basta que sejam seguidos os termos da MPL, que em linhas gerais requerem apenas que os arquivos que contém código do trabalho original estejam sob a licença MPL.

Devido à clareza de seus termos, baseados em definições precisas, a MPL é uma licença mais fácil de entender e aplicar do que a LGPL. Portanto, quando se busca uma licença que combine as vantagens do modelo recíproco com o modelo permissivo, a MPL é uma ótima alternativa.

Porém, uma desvantagem da MPL em relação à LGPL é a incompatibilidade com a GPL. Não é possível juntar dois projetos que estejam sob as licenças MPL e GPL, pois a MPL obriga que o código original mantenha-se como MPL e a GPL obriga que o trabalho como um todo e consequentemente cada uma de suas partes seja GPL. Nesse caso, a alternativa é o licenciamento múltiplo, conforme explicado na MPL.

5.3.3 A Licença Eclipse

Assim como a Netscape, a IBM é uma organização para a qual o software livre tornou-se peça chave do posicionamento estratégico, e neste sentido usou seus recursos para dar um tratamento diferenciado aos aspectos legais envolvendo esse tipo de software. A *Eclipse Public License* (EPL) é a terceira geração de licenças de software livre originadas na empresa.

A primeira delas foi a *IBM Public License* (IPL), publicada em Agosto de 1999 (um ano após a empresa anunciar o suporte ao sistema operacional Linux), marcando o período em que a IBM enfatizava seu apoio à *Open Source Initiative* e o uso do software livre. Um diferencial notável em relação à GPL é colocar a responsabilidade legal no distribuidor do código (e não no desenvolvedor/colaborador), mas a maior fonte de incompatibilidade

com esta são as restrições ligadas a patentes: a licença IPL deixa de fazer efeito em caso de disputas envolvendo patentes.

A *Common Public License* veio a público em Maio de 2001 como uma substituta à IPL. Um dos seus objetivos era encorajar o uso da licença por terceiros, e para isso ela conta com terminologia mais genérica, que não define especificamente a IBM como fornecedor do software original. A CPL se assemelha à GPL no aspecto da distribuição de programas modificados, pois também obriga a disponibilização do código fonte das modificações, mas permite que software não-CPL seja ligado a bibliotecas CPL sem obrigar o software a adotar a mesma licença. Nesse sentido, ela se aproxima mais da LGPL.

Com a criação da *Eclipse Foundation*, uma ONG responsável pelo projeto Eclipse e por cultivar a comunidade e ecossistema em torno dele, fez-se necessária uma licença que não estivesse tão focada em litígios e patentes, e assim surgiu a EPL. Ela ainda é incompatível com a GPL 2.0, em particular por obrigar o distribuidor a licenciar patentes relacionadas a modificações feitas no software original, bem como por incluir um dispositivo que desestimula processos envolvendo patentes, mas é aprovada pela OSI e considerada licença de software livre pela FSF. O uso de bibliotecas EPL em software corporativo é seguro, desde que observada a questão da distribuição de modificações.

Vantagens e Desvantagens

A licença Eclipse é recomendada para projetos que façam parte do ecossistema do Eclipse, que é muito utilizado no desenvolvimento de software em diversas linguagens. Porém, ao desenvolver um projeto desses, é muito importante estar atento aos componentes utilizados, pois essa licença não é compatível com a GPL 2.0, sendo essa sua principal desvantagem.

5.4 Quadro Comparativo

A seguir apresentamos um quadro comparativo entre as licenças mais importantes estudadas neste capítulo, destacando suas principais características de acordo com a nossa avaliação.

	BSD	MIT	Apache	GPL 2.0	GPLv3	AGPL	LGPL	Mozilla / EPL
Reciprocidade[1]	Ausente	Ausente	Ausente	Total	Total	Total	Parcial	Parcial
Clareza / Simplicidade[2]	Média	Alta	Alta	Média	Média	Média	Baixa	Alta
Adoção[3]	Alta	Baixa	Média	Alta	Média	Baixa	Alta	Alta[6]
Compatibilidade com a GPL[4]	Sim	Sim	v3	2.0	v3	Sim[7]	respectiva	Não
Licenças derivadas[5]	Sim	Sim	Sim	Não	Não	Não	Não	Sim

[1] *Ausente*: não exige licenciamento de trabalhos derivados sob os mesmos termos; *parcial*: dispensa o licenciamento equivalente se o projeto original for componente do novo projeto; *total*: obriga qualquer trabalho derivado a ser licenciado sob os mesmos termos.

[2] É necessária atenção especial aos termos omissos e interpretações diversas em licenças de assertividade média ou baixa, observando as restrições delineadas ao longo deste capítulo.

[3] Licenças com maior grau de adoção facilitam a disseminação do projeto entre usuários e empresas e novos desenvolvedores, na medida em que aumentam a chance de contato prévio com as mesmas.

[4] A compatibilidade com essa licença em particular é um critério importante, pois ela é uma das mais usadas em projetos já existentes, e, ao mesmo tempo, uma das que mais apresenta restrições de reciprocidade (sendo a única totalmente recíproca analisada neste capítulo).

[5] Licenças que permitem a criação de derivadas podem oferecer maior liberdade, mas tornam a verificação de compatibilidade menos trivial.

[6] Adoção alta se considerarmos também as licenças derivadas da MPL.

[7] Neste caso prevalece a AGPL.

5.5 Metodologia para Escolha de Licenças

Para finalizar este capítulo, apresentaremos uma metodologia para escolha da licença em um novo projeto de software livre. É importante ressaltar que ela é apenas uma sistematização das principais características das licenças vistas até esse momento e não abrange detalhes de implementações específicas do software, que serão estudadas no capítulo seguinte. Para uma análise mais completa da situação do projeto, é essencial adotar uma postura crítica em relação a todos os aspectos dos componentes envolvidos. Para facilitar essa tarefa, consulte o Capítulo 7, que apresenta algumas ferramentas e processos que podem ser utilizadas para este fim.

A Figura 5.1 representa a metodologia proposta na forma de um fluxograma. O processo é bastante simples e se inicia na decisão de distribuir um programa como software livre. No caso de ser uma contribuição a uma comunidade que já existe, por exemplo, um plugin para o ambiente Eclipse, o ideal é utilizar a mesma licença do projeto principal dessa comunidade, de forma a evitar qualquer incompatibilidade tanto atual como também futura.

Por outro lado, se for um projeto mais independente, não atrelado de nenhuma forma a alguma licença pré-existente, o principal fator de decisão é até que ponto o detentor dos direitos quer impor limitações no seu uso e distribuição. Conforme visto neste capítulo, cada tipo de licença possui vantagens e desvantagens. Se a ideia é que o projeto seja utilizado apenas em software livre, fortalecendo a comunidade, sem se importar com a incompatibilidade com projetos que utilizem outra licença, deve ser utilizada a GPL. A escolha da versão também depende dos propósitos e dos componentes utilizados, lembrando que a versão dois é incompatível com a três. Sendo assim, para obter máxima compatibilidade com outros projetos que também utilizam a GPL, uma solução é fazer o licenciamento na versão "2 ou superior", enquanto que para seguir mais de perto os princípios da *Free Software Foundation* a escolha deveria ser a GPLv3, ou até mesmo a AGPLv3, no caso de um sistema que irá rodar em um servidor para a Internet.

No caso do projeto querer se beneficiar de maior adoção, o ideal é escolher uma licença com menores dificuldades de compatibilização. Porém, se ainda é importante que alterações do próprio projeto sejam retribuídas aos autores, a licença a ser escolhida deve

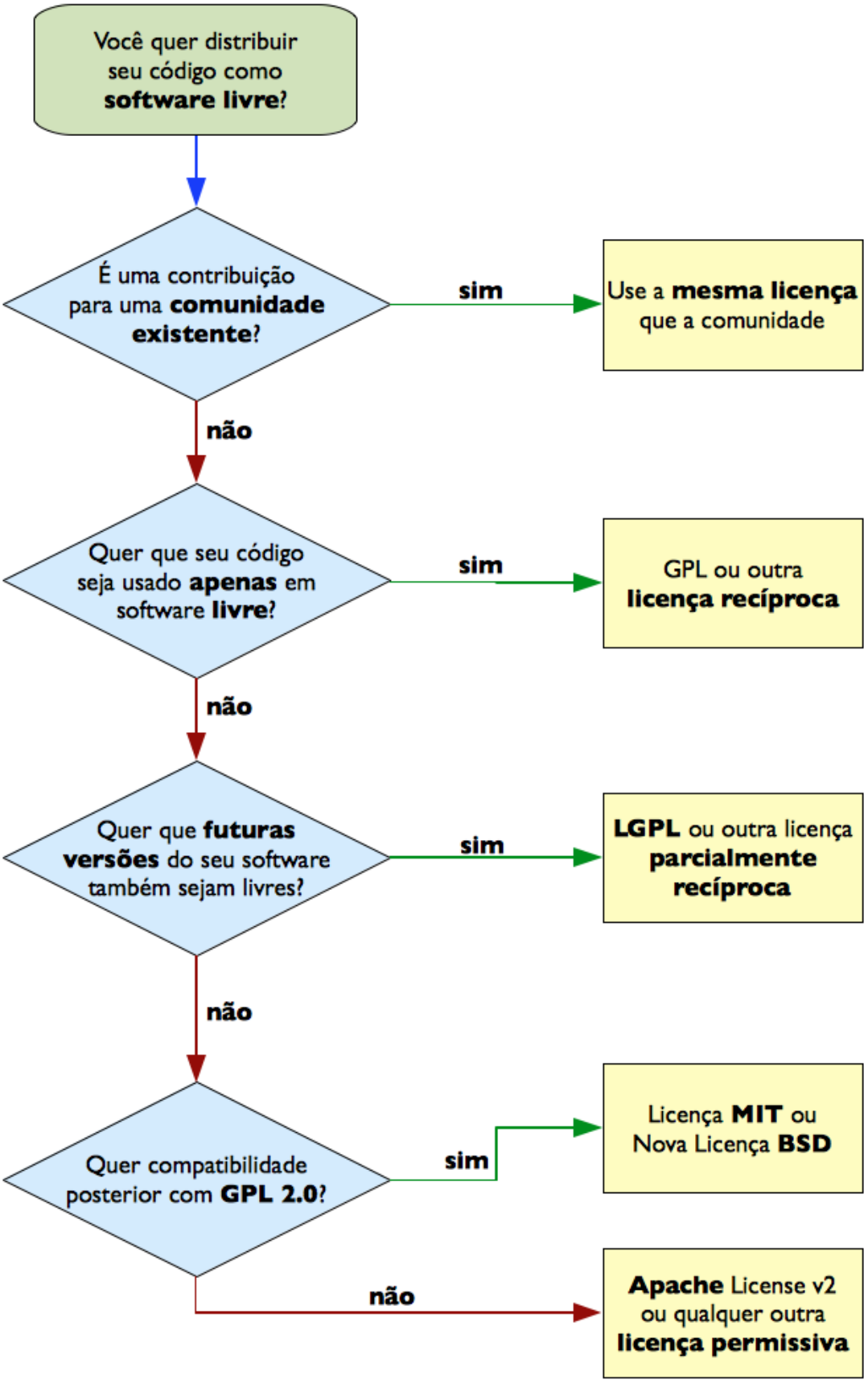


Figura 5.1: *Metodologia para escolha de licenças*

ser uma recíproca parcial, que pode ser a LGPL, para facilitar a compatibilidade com a GPL, ou alguma das outras licenças que seguem esse modelo, como por exemplo a Licença Mozilla, que possui termos bem mais claros.

Se não houver a necessidade de exigir que modificações sigam a mesma licença para sua distribuição, pode ser utilizada qualquer licença permissiva, aumentando ainda mais as chances de adoção do projeto. No caso de ser necessário a compatibilidade com a GPL 2.0, as licenças mais indicadas são a MIT ou Nova BSD, que não fazem nenhum tipo de exigência que não esteja nos termos da GPL, tornando assim possível que o projeto seja relicenciado como GPL posteriormente. Não havendo essa necessidade, também pode ser escolhida a licença Apache, que é um pouco mais formal e adequada, do ponto de vista jurídico, do que as anteriores.

Mas, acima de tudo, o importante é tentar escolher uma licença que já seja conhecida e cujo texto seja verificado por advogados, pois isso facilita sua adoção e evita problemas legais no futuro. A seguir veremos em mais detalhes as questões de compatibilidade entre licenças.

Capítulo 6

Compatibilidade entre Licenças

Gerenciar possíveis conflitos entre licenças é muito importante para não prejudicar a reputação do projeto e evitar complicações legais. Por esse motivo, ao iniciar um novo projeto, os potencias problemas de compatibilidade entre as licenças de componentes que serão utilizados devem ser examinados assim que possível. No caso de projetos comerciais, é recomendável já declarar a licença que será utilizada na especificação do programa. Já em projetos gerenciados por uma comunidade de software livre, a licença deve ser mencionada no acordo de contribuição. Dessa forma, os programadores responsáveis já ficam cientes que devem buscar componentes compatíveis desde o início [Opeb].

Neste capítulo, faremos uma análise da compatibilidade entre licenças, considerando, por um lado, as obrigações que são colocadas nas cláusulas das licenças e, por outro, as diversas formas como componentes de software podem ser utilizados em conjunto e distribuídos. Tudo isso deve ser visto tendo como base toda a legislação que, de alguma forma, cobre o licenciamento de software, cujos principais aspectos foram tratados no Capítulo 4.

6.1 Trabalhos derivados em software

Para analisar as possibilidades de licenciamento de um trabalho que, de alguma forma, é influenciado por um software livre, é necessário, antes de mais nada, definir com clareza qual é a relação estabelecida entre os dois programas. A maioria das criações são influenciadas de alguma forma por trabalhos pré-existentes, porém, se o novo trabalho utiliza muito pouco do original, baseia-se apenas em conteúdos não protegidos, tais como ideias originais do autor, ou faz tantas alterações que o resultado não mais se parece com o trabalho pré-existente, então essa criação deixa de ser vista como um trabalho derivado pela lei, podendo ser considerada como um trabalho original. Nesse caso, o autor é livre para escolher a licença que desejar.

Estudaremos a seguir alguns casos apresentados por Lothar Determann em sua publicação no *Berkeley Technology Law Journal* [Det06], com o objetivo de identificar situações em que a compatibilidade entre as licenças de software possui restrições ou não, de acordo com a classificação do trabalho como derivado ou original.

Determann inicia seu artigo discutindo as possibilidades de trabalho resultante, de acordo com a lei de *copyright* americana, após a combinação entre material novo e pré-existente. São levantadas as seguintes opções:

- trabalho original (não-derivado): quando pouca coisa ou apenas elementos não protegidos do trabalho pré-existente estão presentes no trabalho novo;
- trabalho derivado: se a parte original altera a *substância* do material pré-existente e ambos são criativos;
- compilação: se material pré-existente é arranjado de forma criativa;
- cópia não-literal: se o trabalho novo altera o pré-existente de forma não criativa ou substancial, com o resultado final sendo quase idêntico ao pré-existente;
- cópia literal: material pré-existente arranjado de forma não criativa;
- trabalho não protegido: se a combinação não envolve nenhuma alteração, duplicação ou arranjo criativo.

Conforme pode ser observado pela classificação acima, a criatividade é um fator importante para definir o quão original é um trabalho. No caso dos programas de computador, que possuem uma característica mais utilitária, conforme visto no Capítulo 4, essa questão deve ser analisada sob uma perspectiva diferenciada. Determann afirma que modificações que são ditadas por fatores externos ou considerações sobre funcionalidades, em geral, não afetam a relação entre o autor e seu trabalho criativo, apesar dos direitos sobre adaptação estarem na lista dos direitos protegidos pela lei de *copyright*. Dessa forma, apenas mudanças internas do código, mais difíceis de serem distinguidas pelos usuários, seriam capazes de afetar os direitos de adaptação de um autor. Por outro lado, combinações com programas separáveis ou modificações na interface ditadas por requisitos funcionais, em geral, não deveriam ser consideradas trabalhos derivados criativos [Det06].

Considerando as características únicas do software, Determann apresenta a seguinte definição para determinar a situação de um programa de computador dentro das definições da lei de *copyright*:

> "uma combinação de um programa de computador protegido por *copyright* com outro constitui um trabalho derivado do(s) programa(s) se a combinação é (a) suficientemente permanente, (b) contém porções significativas e criativas do(s) programa(s) (em oposição a informações de interface ou incorporação puramente funcional de código de proteção), (c) é criativo em si mesmo (em oposição a representar a única solução de combinação ou maneira tecnicamente mais eficiente de fazê-la) e (d) envolve mudanças internas criativas e significativas que não podem ser facilmente separadas ou distinguidas do outro programa. Se as porções adaptadas ou mudanças causadas pela combinação não são criativas, mas apenas funcionais em sua natureza, a combinação não

> constitui um trabalho derivado. Nessa caso, se uma combinação de software não envolve essas mudanças internas qualificadas ao(s) programa(s) combinados, ela pode constituir uma compilação (ou cair fora do escopo da lei de *copyright*)" [Det06].

Um dos pontos técnicos ressaltados por Determann é a interoperabilidade e comunicação entre programas diferentes. Ele explica alguns dos mecanismos que podem ser utilizados para realizar a comunicação:

- Sinais: comunicação unidirecional assíncrona entre processos. Um processo envia um sinal para o outro, o sistema operacional gera um evento para a aplicação alvo e esta deve implementar um mecanismo para tratar esse sinal;
- Canais nomeados (em inglês, *named pipes*): através de um *pipeline* pré-definido no sistema, duas aplicações se conectam a um canal nomeado, uma escrevendo enquanto a outra lê os dados colocados nele;
- Arquivos: podem ser utilizados para trocar dados entre aplicações, em formato texto ou binário;
- Memória compartilhada: parte da RAM é compartilhada entre aplicações, que podem escrever e ler dados dela;
- Soquetes: provêem uma API para transporte de dados, normalmente através de um protocolo de rede. Uma aplicação ouve em determinada porta da máquina, enquanto outra, que roda local ou remotamente, se conecta nesse soquete, sendo então possível que ambas aplicações realizem uma comunicação bidirecional.

Em qualquer um desses casos, os programas permanecem distinguíveis do ponto de vista lógico, e a seleção do mecanismo é ditada por requisitos de funcionalidade. Portanto, os programas assim combinados não se tornam automaticamente parte de um trabalho maior derivado, devido à não necessidade de uma criatividade significativa na combinação e da ausência de mudanças internas significativas [Det06].

Determann ilustra sua definição de programa derivado através de alguns exemplos, que veremos a seguir, baseados em casos reais da jurisprudência americana.

O primeiro caso trata de empacotamento de programas com funcionalidade isoladas. A hipótese básica é a de um revendedor que compra software de dois fabricantes diferentes, cujas funcionalidades são semelhantes, porém, independentes e isoladas (por exemplo, dois editores de imagens diferentes). O revendedor pega um CD de cada fabricante e coloca em uma caixa para vendê-los em conjunto, mas é esperado que o consumidor que compra essa caixa instale apenas uma das alternativas em seu computador. Nesse contexto, a combinação dos programas em um pacote é considerada muito separável (portanto não estável) e não suficientemente criativa (original) para qualificar como um trabalho para os propósitos da lei de *copyright* americana. Além disso, os programas não modificam um ao outro durante a execução, nem mesmo temporariamente, o que é outra razão para não

serem considerados como trabalhos derivados. Mesmo que os programas fossem copiados em um mesmo CD, aumentando a estabilidade da composição, esse arranjo físico dos bits de cada programa não envolve a criatividade necessária para considerá-lo um trabalho derivado. Do ponto de vista lógico, esses programas são sempre vistos como separáveis, seja no CD ou no disco rígido do usuário e, portanto, são interpretados como programas distintos.

O próximo caso discutido por Determann trata de modificações da saída na tela através de uma combinação de software. Ele é baseado em um caso real que foi a julgamento, entre MicroStar, Galoob e Midway, explicado em mais detalhes em um artigo de 2003 do *Santa Clara Computer & High Technology Law Journal* [Och04]. A hipótese levantada é de uma empresa A que faz um jogo de computador que consiste em três partes: o *game engine*, uma biblioteca com a arte do jogo e um arquivo de mapa. As pessoas que compram o jogo têm o direito de instalá-lo e jogá-lo, mas não de criar trabalhos derivados. As três partes distintas interagem durante a execução do jogo: o *game engine* utiliza o arquivo de mapa para definir qual imagem aparecerá em cada área da tela e, então, pega o arquivo correspondente da biblioteca e o renderiza no local apropriado. Em um certo momento, duas outras empresas entram no mercado com produtos relacionados ao jogo. A empresa B cria um adendo que permite aumentar a velocidade do jogo, alterar parâmetros tais como força dos personagens ou corrigir erros. Já a empresa C cria arquivos de mapas que contém instruções para combinações alternativas de elementos de cenário que permite a criação de novas fases. Em ambos os casos, os novos programas, isoladamente, não têm qualquer semelhança com o produto da empresa A, tanto no nível do código como no nível do objeto compilado. Cabe aos consumidores comprarem os adendos das empresas B e C e executá-los simultaneamente ao jogo da empresa A. Isso faz com que o computador execute as instruções do jogo de acordo com diretrizes dadas pelos adendos, porém, quando a pessoa fecha os programas, nenhuma alteração permanece na cópia do executável da empresa A instalada no computador do cliente.

O *game engine* e o arquivo de mapa são considerados como trabalhos literários, conforme explicado no Capítulo 4, sendo suficientemente originais para garantir sua proteção. A biblioteca de arte é uma coleção de items arranjados de forma a otimizar seu acesso, e portanto não é suficientemente criativa para constituir um trabalho coletivo ou uma compilação, havendo apenas a proteção dos itens gráficos individualmente. Quando executadas, essas três partes criam sequências do jogo que podem ser protegidas como uma obra audiovisual. Os adendos das empresas B e C não contém nenhuma parte dos trabalhos de A e, portanto, não podem ser considerados trabalhos derivados ou compilações. Porém, as empresas B e C são consideradas responsáveis pelas ações de seus consumidores, pois distribuem seus programas com o propósito exclusivo de combiná-los com o jogo da empresa A. Essa responsabilidade foi reafirmada em dois casos importantes da jurisprudência americana: *A&M Records, Inc. v. Napster, Inc.* (2002) [Uni01] e *Metro-Goldwyn-Mayer Studios, Inc. v. Grokster* (2005) [Sup05], ambos nos quais empresas de programas de compartilhamento ponto-a-ponto foram consideradas responsáveis pelos

arquivos distribuídos entre os usuários de suas redes. Quando os usuários executam os adendos, o jogo é afetado de duas maneiras: como obra audiovisual, determinada pelo que aparece na tela, e como trabalho literário, no conjunto de instruções que compõem o jogo, conforme acordo da Convenção de Berna. Para determinar se isso constitui um trabalho derivado, é necessário observar se a combinação é suficientemente permanente, se contém partes significativas do programa, se é criativa por ela mesma e se envolve mudanças internas significativas e criativas ao outro programa que não podem ser facilmente separáveis do programa original.

Esse caso é analisado por Determann em dois níveis: o que aparece na tela e o código que é executado. Nessa primeira parte, veremos o efeito sobre o que aparece na tela para o usuário. Quanto à permanência, apesar das alterações no jogo causadas pelos adendos serem reversíveis quando os adendos não estão em uso, o caso *MAI Systems Corp. v. Peak Computer, Inc.* (1993) [Cor] abriu uma jurisprudência definindo que cópias em RAM são suficientemente estáveis para serem consideradas uma violação da lei de *copyright* (nesse caso, a Peak Computer, uma empresa de assistência técnica, rodava nos computadores dos clientes um programa de diagnóstico da MAI Systems sem a devida licença para essa "cópia" que era feita durante o serviço). Isso afeta diretamente as novas fases criadas pelo adendo da empresa C, cuja execução sempre cria uma combinação pré-determinada. Já o adendo da empresa B não gera uma alteração que tem a mesma característica de permanência, pois o resultado depende muito das ações tomadas pelo jogador durante sua execução, e, dessa forma, não é qualificado como trabalho derivado. Quanto a conter partes significativas do programa, as imagens que aparecerão na tela do usuário consistem exclusivamente numa combinação dos itens da biblioteca de imagens do jogo da empresa A, os quais são suficientemente criativos para serem protegidos, o que basta para caracterizá-lo como trabalho derivado nesse aspecto. Já quanto à criatividade da combinação, no caso do adendo da empresa B, em que as alterações se limitam a parâmetros do jogo, o critério de criatividade não é atingido. Por outro lado, o rearranjo dos gráficos para criar novas fases obtidas pelo adendo da empresa C são suficientemente criativos. Por fim, temos o critério de mudanças internas significativas. Ambos adendos alteram de alguma forma a sequência de eventos que acontece durante o jogo, porém no caso do produto da empresa B isso não ocorre de maneira significativa. Portanto, do ponto de vista do que aparece na tela, apenas os consumidores do adendo da empresa C estariam criando um trabalho derivado e, portanto, infringindo os direitos do jogo reservados à empresa A.

Analisaremos agora o ponto de vista do código que sustenta as alterações vistas na tela. Quanto à permanência, os adendos são bastante desacoplados do programa do jogo e dependem muito da interação com o usuário, portanto não se caracterizam como uma combinação permanente para os propósitos de constituir um trabalho derivado ou compilação. Porém, se considerarmos o código presente em memória RAM durante a execução como uma combinação permanente, podemos afirmar que esse conjunto do jogo mais os adendos contém partes significativas do programa original, o que seria um ponto a favor

de considerá-lo um trabalho derivado. Quanto à criatividade da combinação, a questão que é apresentada é se existe apenas um mecanismo eficiente para manipular a sequência do jogo, o que significaria que a implementação desse mecanismo no software não é suficientemente criativa. No nível desta análise, não é possível determinar esse ponto. Para finalizar, quanto a mudanças internas significativas, por uma perspectiva lógica, cada um dos programas se mantém como trabalho literário independente, constituindo instruções para o computador arranjadas de forma a gerar as alterações desejadas na saída para a tela. Isso significa que, mesmo que o adendo da empresa C esteja dentro dos critérios para ser considerado uma nova obra para fins de *copyright*, ele não constitui um trabalho derivado. Dessa forma, no máximo os adendos podem ser considerados como compilações, se considerarmos alguma criatividade no ato de combiná-los com o jogo, e isso não estaria afetando os direitos exclusivos de adaptação da empresa A.

O último caso explicado por Determann trata de ligação dinâmica ou estática. O exemplo dado é de uma empresa A que vende um pacote de gerenciamento de folha de pagamentos que consiste em vários programas, incluindo um banco de dados com as informações de recursos humanos e formulários para gerar relatórios de imposto de renda. Esse pacote é distribuído com uma licença que permite sua instalação e execução, mas proíbe a criação de trabalhos derivados. Há ainda uma empresa B que desenvolve um relatório de imposto de renda que substitui o da empresa A. Esse software da empresa B interage com o banco de dados da empresa A, mas possui seu próprio código e interface de usuário. Essa chamada que o software da empresa B faz ao banco de dados da empresa A é realizada através da interface especificada, que inclui funções oferecidas por subprogramas ou bibliotecas do pacote da empresa A, tais como scripts, macros, cálculos matemáticos e conversões de moeda.

A interoperabilidade entre o pacote da empresa A e o programa da empresa B pode ser obtida através de ligação estática ou dinâmica. No caso da estática, pequenos trechos de código da empresa A são inseridos nos programas da empresa B. Já no caso da ligação dinâmica, o programa da empresa B faz chamadas ao sistema que instruem o computador a executar as funções do sistema A e devolver seus dados para continuar a execução de B. Segundo Determann, como o programa de B não interfere na saída do programa de A de nenhuma forma, não se considera que esteja sendo criado um trabalho de imagem ou audiovisual derivado. Porém, no nível do código, há uma combinação entre os programas tanto no caso estático como no caso dinâmico. Assim, novamente devemos observar os critérios citados anteriormente para determinar se essa combinação seria qualificada como um trabalho derivado. A maior diferença entre ligação dinâmica ou estática está na permanência. No caso da estática, parte do código da empresa A aparece dentro do código da empresa B (infringindo também o direito de cópia). Já na dinâmica, a característica da chamada do programa de A pode ser diferente de acordo com o contexto em que o programa de B está sendo executado. Os programas ficam separados na memória RAM e são conectados apenas através de chamadas sequenciais, não se tornando permanentemente parte de um trabalho maior. Além disso, no caso da ligação dinâmica, a parte do

código que precisa ser copiada é apenas algumas linhas representativas da interface, o que não é considerado normalmente como substancial ou criativo. Quanto à criatividade na combinação, como o objetivo é apenas extrair dados de um programa para uso no outro, considera-se que é uma combinação puramente funcional e portanto não suficientemente criativa para constituir um trabalho derivado. Por fim, quanto às mudanças internas, em nenhum dos casos, a princípio, há modificações no programa da empresa A. Porém, no caso da ligação estática, dependendo da forma como as linhas de código do programa A forem retiradas de seu contexto original para serem copiadas no programa B, isso pode ser considerado uma mudança interna no trecho de código que foi copiado. Portanto, de forma geral, uma ligação dinâmica não cria um trabalho derivado e portanto não está sujeita a ser considerada uma infração de *copyright*. O caso da ligação estática já é um pouco mais controverso e sujeito a interpretações quanto à criação de um trabalho derivado, porém, acima de tudo, essa solução normalmente infringe os direitos do autor original ao duplicar o código para o novo programa.

Os exemplos citados, que foram baseados em casos reais, demonstram que o risco de um programa ser considerado trabalho derivado de outro é baixo se forem observados alguns princípios básicos, como falta de permanência ou criatividade. Porém, o detentor dos direitos do software pré-existente pode proibir as combinações contratualmente, mesmo quando elas não constituem um trabalho derivado.

6.2 Compatibilidade de acordo com a categoria da licença

No Capítulo 5, dividimos as licenças em três categorias, de acordo com as restrições de licenciamento na redistribuição do trabalho ou criação de trabalhos derivados: Permissivas, Recíprocas Parciais e Recíprocas Totais. Aqui veremos em maiores detalhes como cada uma dessas categorias influencia na forma como um trabalho pode ser utilizado quando o resultado final estará sob outra licença.

6.2.1 Licenças Permissivas

As licenças permissivas se mostraram as mais fáceis de compatibilizar com outras licenças, na medida em que permitem que trabalhos derivados sejam redistribuídos sob outros termos, até mesmo como software fechado. Dessa forma, ao adotar um software que utiliza uma licença permissiva, as chances de incorrer em problemas de licenciamento são bastante reduzidas. Porém, a liberdade de distribuir um software derivado sob outra licença não implica na possibilidade de usar qualquer uma delas. Cada licença possui termos específicos, que podem causar conflitos com os de outra licença, dependendo de quão restritivo eles são. Isso acontece principalmente quando em uma das pontas temos a GPL, que obriga que o trabalho resultante não apresente nenhuma exigência além daquelas presentes em seus termos. Analisaremos então, caso a caso, a compatibilidade de cada uma das licenças permissivas estudadas com a GPL.

Em primeiro lugar, vimos a licença BSD, que foi apresentada em diversas versões. A versão original da licença, que incluía a chamada "cláusula de propaganda", não é compatível com a GPL, pois a exigência de mencionar uma determinada organização ao falar do produto não faz parte dos termos da GPL. Sem essa cláusula, temos a versão modificada da BSD, a mais usada atualmente, e esta sim é compatível com a GPL, pois nesse caso não temos na BSD nenhuma exigência que não conste nos termos da GPL. Da mesma forma, a licença MIT/X11 também é compatível com a GPL, dado que em essência os seus termos são bastante semelhantes aos da BSD modificada. A MIT/X11 é a licença recomendada pela própria *Free Software Foundation* quando o detentor dos direitos busca uma licença mais permissiva do que as oferecidas por ela. Por outro lado, no caso da licença Apache 2.0 temos um problema maior. Essa licença é bastante detalhada e possui alguns termos que visam garantir maior proteção àqueles que a adotam. Entre eles, há dois termos que a *Free Software Foundation* considera incompatíveis com a GPL 2.0. O primeiro afirma que, no caso de uma organização iniciar um litígio relacionado a infração de patentes no software sendo licenciado, essa organização tem sua licença terminada. O outro problema de compatibilidade visto pela FSF é que, no caso de uma entidade optar por oferecer garantias para o software, ela fica obrigada a indenizar os contribuidores do software em caso de problemas legais decorrentes dessa escolha. Essas duas condições não existem na GPL 2.0 ou anteriores, portanto não seria possível redistribuir um software Apache sob uma licença GPL 2.0 respeitando as obrigações originais de licenciamento. Já na versão 3 da GPL não há mais essa incompatibilidade com a licença Apache, pois o texto foi revisto de forma a adaptar-se a essas condições.

É importante ressaltar que em todos esses casos estamos tratando da situação em que um programa licenciado de forma permissiva será utilizado e redistribuído com uma licença mais restritiva que a dele. O inverso não é possível. Um software com uma licença mais restritiva não poderá ser distribuído com uma licença permissiva a não ser com autorização expressa do(s) autor(es).

6.2.2 Licenças Recíprocas Totais

Do outro lado do espectro de obrigações temos as licenças recíprocas totais. Segundo seus termos, o licenciante fica obrigado a redistribuir o software sempre sob a mesma licença. Segundo Determann [Det06], combinações de programas com software GPL são até certo ponto perigosas para empresas com um modelo de licença fechado se elas também quiserem distribuir o próprio código GPL.

Quando analisamos a compatibilidade com uma licença recíproca, muitas vezes o foco da questão não está tanto nos termos da licença, pois ela dificilmente seria compatível com qualquer outra. A decisão de compatibilidade é tomada julgando até que ponto o uso do software conforme o caso caracterizaria de fato uma infração da licença ou das leis vigentes no local. Assim sendo, a forma de distribuição tem importância fundamental quando estamos tratando de licenças recíprocas.

A lei de direitos autorais brasileira (vide Seção 4.2) é bem mais restritiva do que a

americana. A lei americana está baseada no que é chamado de cinco pilares do *copyright*, que são o direito exclusivo de:

- reproduzir o trabalho;
- preparar trabalhos derivados;
- distribuir cópias do trabalho;
- apresentar o trabalho publicamente;
- mostrar o trabalho publicamente.

Já a lei brasileira (Lei nº 9.610/98), no artigo 29º, provê uma extensa lista de utilizações da obra que dependem da autorização expressa do autor, incluindo até mesmo "quaisquer outras modalidades de utilização existentes ou que venham a ser inventadas" [Con98].

A União Européia possui uma abordagem diferente. Em seu repositório para projetos utilizados em administração pública, o *Open Source Observatory and Repository* (OSOR), há uma seção de FAQ [Opea] em que em uma das perguntas tenta responder sobre o impacto de utilizar um componente GPL em um programa que está sob a EUPL. Para responder, afirmam que não há problema em usar o componente GPL. Quando o código fonte é combinado sem sofrer modificações, cada componente pode manter sua licença, mesmo que, para o usuário final, o trabalho aparente ser uma coisa só. Basta respeitar os *copyrights* individuais e prover as respectivas licenças. Para o caso de ligações (dinâmicas ou estáticas), que segundo a FSF engatilhariam a característica viral da licença, o OSOR defende que software licenciado sob a EUPL está protegido de apropriação e que não há jurisprudência que diga o contrário na Europa. Porém, na dúvida, dizem ainda que o melhor é contar com a ajuda de um especialista na área e pedir para o autor do componente GPL incluir uma exceção em sua licença que permita explicitamente a integração com a EUPL.

As licenças estudadas aqui foram escritas dentro do contexto da lei americana. Portanto, ao interpretar as licenças, é possível justificar, em alguns casos, que mesmo quando algum uso não está explicitamente listado nos seus termos, havia a intenção de autorizá-lo. Por exemplo, a licença Apache não menciona permissão em usar o software, porém, lista várias atividades que podem ser consideradas uma exploração muito maior do direito de autor, e assim fica implícito que o uso também estaria permitido.

Um problema maior ocorre quando a licença apresenta uma definição de termos jurídicos diferente da que é aceita no país. Por exemplo, o *Copyright Act* dos Estados Unidos apresenta a seguinte definição:

> "*A 'derivative work' is a work based upon one or more pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which a work may be recast, transformed, or adapted.*

> *A work consisting of editorial revisions, annotations, elaborations, or other modifications which, as a whole, represent an original work of authorship, is a 'derivative work'"* [U.S76].

Já a GPL na versão 2.0, conforme visto na Seção 5.2, afirma que um trabalho derivado é "*a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language*". Não se pode por contrato de licença alterar uma definição legal, por isso, para a versão mais nova da GPL houve um grande esforço para adaptar toda a terminologia utilizada, preferindo palavras que não causariam conflitos com as leis. Dessa forma, a GPLv3 não fala mais em "trabalho derivado", mas define o que seria um trabalho "baseado" no trabalho anterior (levando em consideração a necessidade de permissão pelas leis de *copyright*) e define também que um "trabalho coberto" significa tanto o programa não modificado como também um trabalho baseado no programa. Por esse motivo, Determann [Det06] alerta que qualquer pessoa que queira distribuir seus programas em conjunto com código GPL precisa examinar atentamente o alcance e consequência das várias condições e restrições da GPL.

A licença GPL afirma explicitamente que o ato de executar o programa não modificado não é restrito, o que está de acordo com a lei de *copyright* americana. A GPL é mais permissiva que a lei de *copyright* na medida em que também permite, sem aplicar as condições da licença, que sejam feitas cópias do programa e que ele seja modificado, desde que essas cópias e modificações sejam apenas para uso privado, ou seja, não sejam distribuídas para terceiros. Portanto, no caso de um uso privado, não há restrições de compatibilidade com outras licenças, já que a GPL isenta qualquer obrigação nesse caso.

Porém, quando estamos tratando da questão de compatibilidade entre licenças, é importante entender as implicações na distribuição do programa. Como a cópia e reprodução de uma obra são ações restritas pela lei de *copyright*, o autor do software tem direito a impor condições especiais. No caso da GPL, a condição mais importante para essa análise de compatibilidade é o fato do programa como um todo ter que ser distribuído sob a mesma licença. É feita uma ressalva na licença esclarecendo que um software GPL pode ser distribuído no mesmo meio que outros programas sem afetar o licenciamento desses últimos, desde que eles sejam de fato independentes do software GPL.

Então há dois pontos críticos para definir a compatibilidade quando uma das partes é GPL: o quanto um software é derivado do outro e como eles são distribuídos. Se um software é razoavelmente independente do outro que está sob a GPL e se eles são distribuídos separadamente, é seguro afirmar que ele não estará sujeito às condições da GPL, e portanto não há preocupações referentes à compatibilidade de licenças.

Como primeiro exemplo, vamos considerar um software de cálculo numérico que tem como uma de suas funcionalidades a geração de gráficos. Para a geração de gráficos, vamos supor que é feita uma chamada a um programa externo, que é GPL, passando como argumentos os dados numéricos para o gráfico. Se o autor do software de cálculo numérico não realizar uma distribuição conjunta do programa para geração de gráficos, apenas indicando em sua documentação que tal programa é necessário para essa funcionalidade e

como obtê-lo, é improvável que ele tenha qualquer problema de licenciamento por escolher algo diferente da GPL para seu software.

Outro caso em que fica ainda mais clara a independência entre os dois programas é quando a funcionalidade que usaria o programa GPL não depende de uma implementação específica. Por exemplo, vamos supor um software que faz persistência em um banco de dados SQL. O autor do software pode ter usado um banco sob a licença GPL (como por exemplo o MySQL na versão *Community Server*), mas tomando cuidado para deixar configurável em seu software parâmetros para acesso ao banco de dados e não utilizando nenhum recurso que seja exclusivo desse banco. Assim, ele pode distribuir seu software e na documentação apenas indicar que será necessário algum banco de dados e explicar onde configurar como será feito o acesso. Nesse caso, não há dúvidas de que ele não está restrito aos termos da GPL.

Por outro lado, há casos em que o software não depende de uma implementação específica, mas ainda assim há restrições no momento da distribuição. Tomemos como exemplo o sistema de mapeamento objeto-relacional do Java. Desde a versão 5 do Java EE, temos o JPA (*Java Persistence API*) em sua especificação. Há várias implementações dessa especificação, que são usadas na forma de bibliotecas. Vamos supor que o desenvolvedor opte por usar uma implementação licenciada sob a GPL. Nesse caso, se não for utilizado um sistema para injeção de dependências, é mais difícil separar o software sendo desenvolvido da biblioteca que ele está usando. Ainda que a chamada das funções seja genérica o suficiente para poder ser utilizada por diferentes implementações, no código fonte estará declarada a importação de pacotes específicos que provêem a funcionalidade da biblioteca. Dessa forma, ao compilar esse código fonte, terá sido gerado um binário que funcionará apenas com a implementação da biblioteca que foi especificada, perdendo um pouco da característica de independência entre os dois programas. Assim, temos um caso mais parecido com o do primeiro exemplo, porém com um agravante: a comunicação entre os dois programas é muito mais próxima. Aqui, o tipo de informação que é compartilhada e a forma como isso ocorre pressupõe uma integração muito maior entre os dois programas. Não é apenas a chamada de um programa externo para realizar determinada tarefa, mas sim a utilização de toda uma estrutura provida pela biblioteca para realizar algo que talvez seja parte fundamental do sistema.

Portanto, mesmo que a biblioteca GPL não seja distribuída junto com o restante do sistema, há o risco do sistema ser considerado "derivado" da biblioteca, e assim estar sujeito aos termos dessa licença, o que o impediria de ser licenciado sob outros termos. Mesmo que isso não aconteça, há também o risco de que um juiz interprete a seção 2b da GPL de forma que as obrigações não se apliquem apenas a trabalhos derivados conforme definidos na lei, mas também a compilações ou outras formas de combinação que poderiam não estar sujeitas aos direitos exclusivos do autor. Porém, Determann [Det06] afirma que, em um caso como esse, de uma interpretação tão ampla da seção 2b da GPL, a parte prejudicada pode contra argumentar levantando alguns pontos como mal uso do direito autoral, lei da concorrência ou contrato desleal. Por esse motivo, a interpretação mais

restrita da seção 2b é mais provável e apropriada, o que faria com que uma ligação dinâmica com um programa GPL não exigisse o licenciamento sob os mesmos termos, mesmo se ambos os programas forem distribuídos conjuntamente. Porém, Determann não descarta a possibilidade da interpretação mais ampla, mesmo que com ela surjam mais questionamentos sobre a validade da GPL nos contextos citados anteriormente (mal uso do direito autoral, lei da concorrência ou contrato desleal). Em uma situação em que essa interpretação prevaleça sem as correspondentes preocupações quanto à sua validade, a indústria do software poderia sofrer um grande impacto na interoperabilidade, devido ao perigo que essa situação representa na combinação entre programas.

Um caso concreto que merece destaque é o do núcleo do Linux, que será visto com maiores detalhes no Capítulo 8. O núcleo do Linux é distribuído sob a licença GPL 2.0 e ele interage com diversos *drivers* de forma a possibilitar o correto funcionamento do hardware do usuário. Alguns desses *drivers* foram feitos pela própria fabricante do hardware e estão disponíveis como software fechado. Esses *drivers*, para seu funcionamento, interagem com o núcleo, muitas vezes executando algumas de suas rotinas. A dependência entre o *driver* e o núcleo faz com que alguns considerem o *driver* como uma extensão do núcleo, o que obrigaria a adoção da licença GPL. Porém, Linus Torvalds afirmou desde o princípio que permitiria que os *drivers* utilizassem outras licenças, assim, como principal autor do núcleo do Linux, a posição de permitir *drivers* fechados acaba prevalecendo na comunidade, pois o contrato é interpretado com a apuração da vontade das partes. Tanto ele como outros mantenedores do núcleo já deixaram clara sua posição e relação com a *Free Software Foundation* na *Linux Kernel Mailing List*, afirmando também que não irão adotar a GPLv3 por serem contra restrições adicionais impostas nessa versão da licença [Bot06], como por exemplo a necessidade de fabricantes de hardware que incorporam software GPL fornecerem as chaves necessárias para modificar o software executado no hardware.

Em casos como esse, em que a *Free Software Foundation*, autora da licença GPL, considera que o trabalho é derivado e portanto deve ser licenciado também como GPL, mas os detentores dos direitos manifestam uma opinião contrária, favorecendo um uso mais aberto de seu trabalho, o risco de deixar um programa relacionado sob uma licença diferente da GPL é reduzido, pois o posicionamento dos detentores dos direitos é mais importante. Além disso, de acordo com Determann [Det06], a distribuição de programas *add-on* separados deve ser permitida, mesmo nos casos em que a combinação resulta na criação de um trabalho derivado pela definição da lei, devido a forma como a GPL permite explicitamente que os usuários combinem código GPL com qualquer outro código. Nesse caso, o problema só ocorreria na distribuição conjunta do software fechado com o programa GPL, ou seja, a distribuição pode ser feita separadamente e o usuário final pode combinar novamente o programa original com o *add-on*.

Uma forma do autor do programa oficializar as permissões que de outra forma estariam restritas pela GPL, e assim permitir maior compatibilidade com outras licenças, é adicionado *exceções*, como foi mencionado anteriormente. Uma versão da GPL bastante

usada é a *Classpath Exception*, que inclui os seguintes parágrafos [GNU]:

```
Linking this library statically or dynamically with other modules is
making a combined work based on this library. Thus, the terms and
conditions of the GNU General Public License cover the whole
combination.

As a special exception, the copyright holders of this library give you
permission to link this library with independent modules to produce an
executable, regardless of the license terms of these independent
modules,  and to copy and distribute the resulting executable under
terms of your choice, provided that you also meet, for each linked
independent module,  the terms and conditions of the license of that
module. An independent  module is a module which is not derived from or
based on this library. If you modify this library, you may extend this
exception to your version of the library, but you are not obligated to
do so. If you do not wish to do  so, delete this exception statement
from your version.
```

Essa versão da licença foi criada originalmente pela própria *Free Software Foundation* para ser usada com a linguagem de programação Java, com o intuito de viabilizar uma implementação livre de todas as classes da API do Java sem que fosse obrigatório que seus usuários também licenciassem seus programas como GPL. Ainda assim, havia quem considerasse a licença confusa, em particular porque a terminologia da GPL 2.0 era um tanto controversa em relação ao significado de "trabalho derivado". Antes disso, até novembro de 2006, o Java possuia uma licença bastante restritiva, que não permitia sua redistribuição. Por esse motivo, não podia fazer parte de uma distribuição de Linux, por exemplo, e os usuários precisavam fazer seu download manualmente. Dessa forma, a alteração da licença promoveu uma facilidade de distribuição e reuso do sistema [DPGGA10]. Há várias outras licenças com exceções similares, que de forma geral são chamadas de *GPL linking exception*. Alguns exemplos de projetos que as utilizam são o GNU Guile (interpretador e máquina virtual para a linguagem Scheme), as bibliotecas de tempo de execução do GNAT (compilador para a linguagem Ada), a libgcc do GNU Compiler Collection e o compilador Free Pascal, entre outros.

O uso de exceções é um dos padrões levantados por German e Hassan. Em seu artigo [GH09], eles apresentam como vantagem do método a possibilidade de facilitar a integração com módulos sob licenças incompatíveis sem a necessidade de modificar a licença original, o que levaria a proliferação de licenças, discutida do capítulo anterior. Por outro lado, as desvantagens são a necessidade de todos os detentores dos direitos consentirem com a exceção e a possibilidade da forma como o texto for fraseado, se não for estudada cautelosamente, causar algum tipo de problema legal não intencional.

Por fim, vale lembrar da licença AGPL, discutida na Seção 5.2, que amplia o escopo da GPL para programas que são distribuídos via rede. Nesse caso, fica muito difícil evitar a cláusula que obriga que seja mantida a opção do usuário baixar o código fonte do

programa. O simples fato do programa estar hospedado em algum lugar da internet já faria com que isso fosse considerado distribuição do mesmo, o que dificulta ainda mais a compatibilidade com outras licenças. Por esse motivo, alguns advogados recomendam evitar software AGPL por empresas de serviços via Internet que não tenham interesse em distribuir seu código fonte [Boe10].

6.2.3 Licenças Recíprocas Parciais

Licenças recíprocas parciais buscam criar um equilíbrio entre as características permissivas e as recíprocas. Os componentes cobertos e aqueles diretamente derivados sempre mantêm a licença original, mas uma aplicação que combina esses componentes com outros pode ser licenciada, como um todo, sob qualquer licença [Opeb].

As licenças recíprocas parciais variam amplamente, algumas deixando bem mais claro do que outras os casos em que é permitido utilizar outra licença no trabalho combinado. Vale voltar na Seção 5.3 para recapitular em mais detalhes os termos de cada uma das licenças desse tipo estudadas.

Na *Mozilla Public License* (MPL), por exemplo, fica claro em que casos ela se comporta como uma licença permissiva e quando ela se comporta como uma licença recíproca. Então, para cada caso, valem as considerações sobre compatibilidade já expostas neste capítulo. Em relação à GPL, segundo a *Free Software Foundation* [Fre], as licenças não são compatíveis, porém, a versão 1.1 da MPL permite a escolha de uma licença alternativa, o que viabilizaria que módulos MPL e GPL fossem distribuídos conjuntamente em uma aplicação.

A licença recíproca parcial mais fácil de compatibilizar com a GPL é a LGPL, que afirma explicitamente que qualquer trabalho modificado pode ser distribuído sob a GPL. Por outro lado, a compatibilidade com outras licenças de software livre ou com software fechado requer mais atenção, pois suas cláusulas não são tão simples quanto às da MPL.

Há bastante controvérsia em relação ao uso da LGPL para bibliotecas de linguagens orientadas a objetos, como o Java. No site do projeto GNU há um texto de David Turner que visa esclarecer a questão [Tur04], afirmando que o mecanismo de *import* do Java não invalida as exceções feitas na LGPL que permitem distribuir o trabalho sobre outra licença. Resumidamente, o texto explica o seguinte: a posição da *Free Software Foundation* quanto à ligação dinâmica é que seu uso cria um trabalho derivado, caindo no critério de reciprocidade da GPL. A LGPL foi criada para que bibliotecas livres pudessem ser utilizadas em software fechado. Segundo Turner, detalhes de implementação das linguagens de programação não afetam as cláusulas que dão a liberdade de escolha da licença. Apenas é necessário que a aplicação final seja distribuída de tal forma que seja permitido trocar a biblioteca LGPL por uma versão mais nova, e que seja possível depurar essa integração por engenharia reversa. Como as bibliotecas do Java costumam ser distribuídas em arquivos .jar separados, é muito fácil fazer a atualização. É responsabilidade de quem desenvolve a biblioteca não quebrar sua interface, para manter a compatibilidade entre as versões e garantir que os programas que a utilizam continuem funcionando com as ver-

sões mais novas. Além disso, se os arquivos da biblioteca estiverem incluídos na aplicação distribuída, é necessário incluir o código fonte da biblioteca LPGL. Por fim, Turner esclarece que o mecanismo de herança de um linguagem orientada a objetos, como o Java, é equivalente, do ponto de vista de criação de trabalhos derivados, ao mecanismo de ligação tradicional de linguagens como o C, e que a LGPL permite esse tipo de trabalho derivado da mesma forma que permite a chamada ordinária de funções. Por outro lado, como há bastante discussão em torno da questão, há quem prefira alternativas como o uso da GPL com uma exceção clara e bem definida, como a *Classpath Exception* apresentada na seção anterior [Wil09].

Já quanto ao tipo de ligação que o código faz, o caso da ligação estática apresenta maiores dificuldades com a LGPL devido à cláusula que obriga que seja possível para o usuário trocar a versão da biblioteca. Se a biblioteca está incluída no arquivo executável estaticamente, é necessário que o autor do software forneça alguns arquivos adicionais (por exemplo um .o, no caso do C) que permitam que o usuário crie um novo executável com a outra versão da biblioteca. Mas essa cláusula só é um problema em relação a software fechado, a compatibilidade com outras licenças de software livre não é afetada, já que quando o código está disponível o usuário tem tudo o que precisa para trocar a versão de qualquer componente, exceto em casos muito peculiares, como quando o hardware requer uma assinatura digital, conforme discutido na Seção 5.2, sobre GPLv3.

Dessa forma, para o autor de uma biblioteca licenciada sob a LGPL é fundamental fazer um planejamento cuidadoso da interface que fará a comunicação entre a biblioteca e o aplicativo. Essa interface deve seguir os princípios de um bom projeto de software e isolar o máximo possível sua implementação, evitando assim que, durante sua evolução, ocorra um problema de compatibilidade reversa que dificulte a atualização da versão por parte dos programas que usam a biblioteca.

Vale lembrar que, como a LGPL é uma licença bastante complexa, ao distribuir um software coberto por ela sob outra licença é importante ler atentamente todos os seus termos para verificar em qual caso a situação se encaixa e como é possível proceder sem violar a licença. Conforme está escrito no próprio texto da LGPL versão 2.1, contradições entre ela e a licença de outras bibliotecas que podem estar sendo usadas no sistema implicam em não ser possível utilizar ambas no executável que será distribuído.

No próximo capítulo serão apresentadas algumas ferramentas que visam auxiliar nessa difícil tarefa de escolher uma licença, evitando problemas de compatibilidade.

Capítulo 7

Ferramentas e Metodologias para Análise de Licenças

O estudo das licenças envolvidas em um projeto não é uma tarefa executada em apenas um momento. À medida que o projeto evolui, é possível que ocorram alterações na sua licença ou na de componentes que são utlizados por ele [DPGGA10]. Em alguns casos, uma pequena alteração da licença de um componente pode fazer com que seus usuários não possam mais utilizar as novas versões em seus projetos. Um exemplo disso é o caso do IPFilter, cujo autor, em 2001, adicionou uma senteça na tentativa de esclarecer os termos da licença e, como consequência, o projeto OpenBSD teve que trocar seu software de *firewall*, pois a nova licença não permitia a distribuição do software modificado, ferindo os princípios do software livre e tornando-o incompatível com a licença do OpenBSD. Posteriormente, o autor do IPFilter retirou a sentença que na época causou o problema, porém, a alteração no OpenBSD já havia sido realizada e o IPFilter perdeu seu espaço. Portanto, é necessário estar sempre atento para que nenhum dos desenvolvedores inclua código que irá infringir os termos de alguma licença do projeto.

A complexidade de licenças de um projeto grande pode ser bastante alta. Numa distribuição Linux (por exemplo, a Debian Etch) há cerca de 20 mil pacotes e são encontradas neles mais de 300 mil declarações de licença [Gob08], que vão desde simples referências, tais como "este software está licenciado sob a GPLv3" até textos completos da licença e ainda alguns casos de negação, como "este software não está sob a GPL".

Um desafio normalmente encontrado por quem coordena o desenvolvimento em um projeto que utiliza diversos componentes é ter visibilidade e controle sobre o que está sendo utilizado pela equipe de programadores, principalmente no caso de projetos de software livre, em que os processos tendem a ser menos rígidos devido à forma como a comunidade se organiza [Ray01]. Algumas ferramentas foram criadas para facilitar a tarefa de analisar as licenças envolvidas em um projeto ou determinar quais seriam as opções de escolha de licença em um determinado caso. Com isso, a ideia é minimizar erros numa verificação manual das licenças, além de poupar tempo e dinheiro que seriam gastos no caso de um litígio decorrente de uma cláusula de licença que não foi seguida corretamente. Apresentaremos a seguir uma seleção dessas ferramentas, que podem ser

utilizadas isoladamente ou em conjunto.

7.1 FOSSology

O projeto *FOSSology* (www.fossology.org) foi iniciado pela empresa Hewlett-Packard como parte do seu processo interno de governança em TI. Seu objetivo é criar ferramentas para facilitar o estudo e análise de software livre, ajudando empresas a adotá-lo com confiança [Hew].

A ideia começou em 2001, quando a HP criou a *Open Source Review Board* para analisar a proposta de cada software livre liberado pela empresa, o que incluia software que rodava dentro dos equipamentos de hardware fabricados (impressoras, TVs, scanners etc.), distribuições Linux e outros projetos de software livre em que havia participação da empresa. Esse grupo criado assegurava que a HP estava honrando a licença de todos os projetos e que qualquer liberação de propriedade intelectual da HP estava sendo feita de forma consciente e com o nível apropriado de aprovação da companhia [Gob08].

No início, o processo para aprovação de um projeto começava com um formulário em que eram listados todos os software livres utilizados. Nesse primeiro passo já ocorriam três dificuldades: resolução de sinônimos nos nomes do projeto, associação de um nome a um conjunto de código específico e incompletude da informação fornecida pelo usuário [Gob08]. Buscando uma maneira de evitar esses problemas, funcionários da HP iniciaram tentativas de automatizar esse processo, trocando o formulário por um repositório que continha todo o código e analisando as principais informações a partir dele. O primeiro passo foi criar o *Nomos* – nome de um personagem da mitologia grega que representava a lei – que buscava automaticamente as licenças em um repositório de código. A próxima geração dessa ferramenta já se tornou o que é hoje conhecido como FOSSology. Em dezembro de 2007, a HP liberou o FOSSology sob a licença GPL e criou o site FOSSology.org para reunir a comunidade em torno do seu desenvolvimento [Gob].

O FOSSology foi escrito nas linguagens C e PHP e é compatível com a maioria das plataformas Linux. Ele provê uma ferramenta bastante completa para análise de licenças. Ao ser instalado, ele cria um repositório para os arquivos e um banco de dados para gravação e recuperação de meta dados. A partir daí, ele fornece uma interface via web e linha de comando para popular o repositório, executar as principais ações e consultar os relatórios. O projeto possui uma arquitetura aberta e modular para análise de software, incluindo módulos de análise de licenças, extração de meta dados e identificação de tipo MIME. Isso permite que ele analise todos os arquivos presentes em um pacote, ao invés de simplesmente afirmar superficialmente que o pacote usa determinada licença, o que é importante pois o pacote pode ser derivado de outros que usam licenças diferentes e isso pode estar mal documentado.

Os agentes que realizam a análise de licenças usam uma heurística para tentar identificar as licenças presentes no pacote. Porém, esse é um problema complexo, então não há 100% de garantia no resultado. Muitas vezes o autor altera o texto de uma licença

conhecida, criando uma licença personalizada, e então o agente a identifica como a licença conhecida, mas com um menor percentual de ajuste. Também podem existir casos em que o programa não sabe como desempacotar o arquivo para ler as licenças que estão dentro dele, e portanto ele não será analisado. Portanto, o resultados apresentados pelo FOSSology são um bom palpite sobre as licenças contidas no projeto, mas não podem ser considerados autoritativos. Ou seja, o código faz o melhor que pode de maneira automatizada, mas é importante lembrar que as decisões legais devem ser tomadas por advogados [Hew].

Apesar do propósito principal do repositório ser a análise de licenças, podem ser criados agentes para outros usos. Robert Gobeille, criador do projeto, sugere alguns deles [Gob08]:

- identificar código reutilizado;
- identificar projetos dentro de uma distribuição Linux que não estão dentro das diretrizes de licenciamento definidas por ela;
- rastrear vulnerabilidades;
- armazenar meta dados dos arquivos.

Uma versão de demonstração de um repositório do FOSSology está disponível em repo.fossology.org, onde pode ser encontradas, por exemplo, informações sobre a distribuição de Linux Ubuntu, versão 10.04.

7.2 Licensator

O *Licensator* (licensator.appspot.com) [Mar] foi desenvolvido pelo brasileiro Daniel Martins. É um aplicativo web que roda no Google AppEngine e seu código, escrito na linguagem Clojure, está disponível no Github (github.com/danielfm/licensator). Segundo o criador, o aplicativo foi motivado por existirem centenas de licenças de software livre, sendo que a maioria delas ou não é usada mais, ou foi substituída por outra, ou é incompatível com outras licenças. Sendo assim, seu objetivo é ajudar as pessoas a decidirem por uma licença mais adequada, através de um questionário que levanta as informações básicas, fornecendo explicações sobre cada item em uma linguagem acessível para leigos.

As perguntas são as seguintes:

- "Se o seu trabalho é baseado em outros projetos de software livre, quais são as licenças desses projetos?
- Você quer que a licença inclua termos de *copyright* explicitamente?
- Você quer que a licença inclua termos relacionados a patentes explicitamente?
- Você permite que seu trabalho seja usado por software fechado?

- Se o seu trabalho é usado em um serviço de rede, é necessário que o servidor em que ele roda forneça um *link* público para download do código do programa?
- Você quer que trabalhos derivados sejam distribuídos sob a mesma licença?
- As pessoas podem distribuir seu trabalho e cobrar pelo seu uso?"

Após respondidas as sete questões, o aplicativo processa as informações e direciona o usuário para uma página que lista as licenças recomendadas, com um *link* que fornece informações adicionais sobre cada uma delas, incluindo o *link* para o texto completo no site da *Open Source Initiative* e uma lista de licenças compatíveis. Não sendo possível encontrar uma licença compatível com todos os requisitos informados pelo usuário, o sistema sugere que o usuário volte para a página anterior e revise suas escolhas.

Algumas das perguntas estão formuladas de forma muito simplificada e podem induzir o usuário ao erro, como por exemplo a última, que trata da distribuição e cobrança do trabalho. Se a resposta é "não", o aplicativo recomenda alguma licença como a GPL, quando na verdade a GPL é bastante clara quanto a permitir a cobrança, como pode ser observado na seguinte frase da Seção 4 da última versão da licença: " *You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee*". Provavelmente o autor do aplicativo fez essa escolha considerando que a GPL também obriga que o código fonte seja distribuído com o binário sem cobranças adicionais, e que ele pode ser usado e redistribuído livremente. Isso implica que mesmo que o autor original cobre pelo arquivo executável, nada impede que seu cliente revenda para outros por um preço menor ou até o forneça gratuitamente, o que tornaria muito difícil manter um modelo de negócio sustentável baseado somente na venda do programa. Porém, impedir o uso comercial iria contra tanto a liberdade 1 da definição de software livre, proposta pela *Free Software Foundation* como também o item 6 da definição de código aberto da *Open Source Initiative* (vide Capítulo 2).

7.3 OSOR.EU License Wizard

O *License Wizard* da OSOR.EU (www.osor.eu/legal-questions-1/licence-wizard) é um guia para auxiliar a busca da licença mais adequada [OSO]. Seu funcionamento é bastante simples: o usuário navega através de uma rede de páginas HTML selecionando *links* que correspondem às respostas para algumas perguntas determinadas, até chegar num ponto final do processo, com uma recomendação de licença.

O início se dá através de um levantamento da situação da propriedade intelectual do projeto: o usuário desenvolveu o software do zero, combinou componentes pré-existentes para criar um novo trabalho, ou apenas modificou um programa de outra pessoa? A partir daí as opções se ramificam.

As perguntas e respostas são todas bastante detalhadas e explicadas, algumas vezes havendo observações adicionais ou páginas intermediárias apenas para esclarecer alguns pontos, tornando todo o processo bastante elucidativo quanto às questões relacionadas ao

licenciamento de software livre. A cada passo, a próxima página faz uma breve descrição da situação já definida, e a resposta final ainda inclui uma descrição das obrigações do usuário, tais como, por exemplo, manter as informações de *copyright* e licença de componentes utilizados no projeto. Por exemplo, se o usuário escolher como primeira opção que está modificando um software recebido de outra pessoa e no segundo passo afirmar que o software original é MPL, a resposta final explica que um binário poderá ser distribuído sob qualquer licença, mas que o código fonte deve necessariamente manter a licença MPL.

O *License Wizard* pode ser considerado conservador em suas alternativas, portanto, diminui os riscos de problemas referentes a um licenciamento inadequado. Em casos em que pode haver alguma dúvida quanto a interpretação, a resposta começa com a alternativa mais segura, e depois explica em que casos poderia ser feita uma exceção. Por exemplo, se o usuário for utilizar um componente GPL como parte de um trabalho maior, fica claro que ele deve distribuir seu programa como GPL, porém, no final da página há uma explicação sobre a noção de trabalho derivado, que depende da lei de *copyright* que será aplicada, afirmando que em alguns tipos de uso normalmente não se considera que sejam criados trabalhos derivados, e nesse caso cada componente poderia ser licenciado separadamente [OSO].

7.4 Carneades

O *Carneades* (carneades.berlios.de) é uma aplicação para mapeamento e avaliação de argumentos, além de também ser disponibilizado como uma biblioteca para construir aplicações que suportam vários tipos de tarefas de argumentação [Gor]. Ele vem sendo desenvolvido no *Fraunhofer Institute for Open Communication Systems* (FOKUS), da Alemanha, desde 2006, e conta com a colaboração de pesquisadores de universidades da Inglaterra, Holanda, Estados Unidos e Canadá. Ele é baseado em um modelo matemático da filosofia de argumentação de Doug Walton, que também é um dos participantes do projeto.

A versão mais recente foi implementada utilizando o paradigma de programação funcional, através da linguagem Clojure. O código está em um repositório Git e está licenciado sob a EUPL v1.1.

O líder do projeto, Thomas Gordon, é membro do projeto Qualipso e sugere o uso do Carneades para ajudar desenvolvedores a construir, explorar e comparar teorias legais na análise de licenças de software livre em casos particulares [Gor10]. Foi construído um protótipo que parte do princípio que resolver questões de licenciamento é um processo argumentativo em que teorias alternativas de conceitos da lei de *copyright*, como o conceito de trabalho derivado, em conjunto com fatos de casos particulares, são construídos e avaliados criticamente. Foi criada uma ontologia de licenças de software livre utilizando a linguagem OWL (*Web Ontology Language*), que foi utilizada para modelar diversas licenças, incluindo a Apache 2.0, BSD, MIT, MPL, EPL e várias variantes da GPL. Além

disso, também foi criada uma ontologia para descrever projetos de software, incluindo várias relações entre componentes usados em um projeto, no nível de abstração necessário para analisar questões de licenciamento. Conceitos legais relevantes foram modelados usando regras de inferência revisáveis no formato *Legal Knowledge Interchange Format* (LKIF). Através dessa base de conhecimento são construídos e avaliados os argumentos a favor e contra referentes a decisão de se uma licença em particular pode ser usada em um determinado projeto. Por outro lado, fica claro que a resolução de compatibilidade entre licenças é um processo que não é bem definido o suficiente para poder ser totalmente automatizado, pois no âmbito legal nunca existe uma única resposta correta [Gor10].

Thomas Gordon levanta os seguintes casos de uso que devem ser incluídos numa ferramenta que visa auxiliar usuários a construirem, avaliarem e visualizarem argumentos relacionados a compatibilidade entre licenças de software livre [Gor10]:

- "declarar um vocabulário para sistemas de software, licenças e conceitos de *copyright*, incluindo relacionamentos entre trabalhos, tais como um ser derivado do outro;
- representar regras de lei de *copyright* (considerando todas as complexidades referentes a leis, tais como interpretações alternativas, exceções e níveis de aplicação) e partes do domínio de engenharia de software relevantes para analisar a compatibilidade de licenças;
- definir formalmente os termos e condições dos tipos de licença mais comuns;
- descrever, em linguagem formal, os fatos de uso e relações derivadas entre trabalhos de software;
- representar evidências sobre os fatos materiais;
- construir argumentos legais a partir das regras legais relacionadas ao caso;
- avaliar um conjunto de argumentos, considerando provas aplicáveis e hipóteses sobre as crenças das pessoas responsáveis por tomar a decisão;
- usar esquemas de argumentação para revelar premissas ocultas de argumentos e formular perguntas críticas;
- visualizar relações entre conjuntos de argumentos, de forma a obter um resumo compreensível;
- determinar o conjunto mínimo e consistente de premissas que, se aceito como verdadeiro pela audiência responsável, seria suficiente para provar ou refutar uma dada declaração, dependendo de um dado objetivo."

Como projeto exemplo para análise de licenças foi utilizado o próprio Carneades, em que o compilador do Clojure utiliza a licença EPL, as bibliotecas gráficas utilizam licença BSD, o Pellet está sob a AGPL e a biblioteca para OWL está sob LGPL. Modelar todas essas relações dentro da definição da ontologia ainda é um trabalho complexo e manual

de interpretação do sistema e dos termos da licença. Há necessidade de conhecimento avançado sobre ontologias, lógicas não-monotônicas e outros conceitos relacionados. Para declarar as regras em LKIF é usado o formato XML, em um esquema um tanto verboso e de baixa legibilidade. Por isso, no estado atual essa é uma ferramenta mais de interesse acadêmico do que de uso geral por desenvolvedores de software. Além disso, apesar do Carneades ser uma ferramenta livre, a ontologia e base de regras para análise de licenças desse projeto exemplo ainda está em desenvolvimento e não foi publicada.

7.5 Open Source License Checker

O *Open Source License Checker* (forge.ow2.org/projects/oslcv3) é uma ferramenta do *OW2 Consortium*, escrita em Java, para gerenciar os riscos associados a licenças de software livre [OW2].

O OW2 é um consórcio de software livre independente, aberto a empresas, organizações públicas, universidades e indivíduos, cuja missão é promover o desenvolvimento de middleware, aplicações de negócio e plataformas de computação elástica.

O sistema permite a importação de projetos através de diretórios, arquivos compactados ou diretamente de um sistema de controle de versão do tipo CVS ou SVN. Ele consegue identificar código fonte em Java, Javascript, PHP, Python e C/C++. Ele é capaz de encontrar licenças similares em comentários do código fonte e em arquivos texto comumente utilizados para esse fim. O algoritmo utilizado para identificar a licença é baseado no trabalho de Paul Heckel [Hec78]. Além disso, o sistema identifica algumas palavras "perigosas", como "*shareware*" e "patenteado", que representam informações adicionais significativas na licença.

O sistema inclui um repositório com o texto completo das licenças mais comuns e seus meta dados: nome da licença, lista de licenças compatíveis e lista de tags (nome de autor, ano etc.) encontradas no texto original da licença.

A ferramenta é capaz de identificar conflitos locais de licença, ou seja, referências do tipo um arquivo A não poder importar ou incluir um arquivo B devido a restrições de licença de um para o outro (por exemplo, um arquivo sob a GPL não pode importar um arquivo sob a licença do PHP). Está previsto que futuramente a ferramenta consiga identificar também conflitos globais dentro do projeto, que não são gerados por uma referência direta entre dois arquivos. Outras funcionalidades interessantes que já são suportadas são mostrar as tags encontradas, identificação de exceções de licenças e listagem de licenças compatíveis com um determinado pacote.

O *Open Source License Checker* possui uma interface gráfica e também pode ser operado pela linha de comando. Os resultados das análises podem ser exportados em formato PDF ou RTF.

Apesar da proposta ser boa e de fácil utilização, a implementação ainda precisa ser melhorada. Em teste realizado com a versão do projeto MemeThis [1] hospedada no Source-

[1] http://memethis.com

forge, cujos comentários de código fonte deixam perfeitamente claro para um ser humano quem são os detentores dos direitos e qual é a licença utilizada, a ferramenta foi capaz de identificar apenas uma pequena parcela dessas informações, referente a algumas das bibliotecas que foram utilizadas, e não as dos arquivos principais.

7.6 Black Duck

A BlackDuck (www.blackducksoftware.com) é a empresa mais conhecida que explora comercialmente o serviço de análise de licenças. Frequentemente ela promove seminários via web com temas sobre os aspectos legais do software livre, gerenciamento de riscos associados ao seu uso em empresas e temas relacionados. Já são mais de 15 seminários disponíveis online em seu site [Blab]. Os serviços pagos oferecidos incluem treinamentos e consultoria sobre licenças de software livre.

Além disso, a empresa oferece o produto *Black Duck Suite*, que inclui os produtos *Back Duck Code Center*, *Export* e *Protex*, além de um *Software Development Kit* (SDK) que permite a integração com algumas ferramentas já conhecidas de desenvolvimento. A proposta do *Black Duck Suite* é automatizar processos chave relacionados ao gerenciamento de código de software livre durante o ciclo de vida da aplicação. O *Protex*, em particular, é o módulo responsável por analisar a propriedade intelectual do código e as obrigações decorrentes das licenças utilizadas no projeto, validando se não há ocorrência de nenhuma infração. Havendo qualquer discrepância, ela é destacada e a ferramenta permite que seja feito seu acompanhamento até chegar a uma eventual solução. De acordo com o site da BlackDuck, o seu grande diferencial é possuir a base de conhecimento de código de software livre mais abrangente do mundo, incluindo mais de 350 mil projetos de mais de cinco mil sites, com mais de duas mil licenças de software únicas. Isso permite que um código seja reconhecido mesmo que as informações de *copyright* e licença tenham sido eliminadas, identificando possíveis usos não autorizados de código encontrado na Internet. Além disso, o *Black Duck Code Center* é capaz de utilizar essa mesma base para encontrar um componente adequado para o projeto, tanto do ponto de vista de licença como também outras características, tais como vulnerabilidades conhecidas.

As principais vantagens do uso de seus produtos destacadas pela Black Duck são:

- reduzir os riscos de negócio, guiando os funcionários no uso diário de componentes de software licenciados;
- acelerar o desenvolvimento de software, permitindo que desenvolvedores reutilizem componentes de qualidade tanto de software livre como também comerciais em larga escala;
- minimizar os custos do negócio, através de um gerenciamento mais pró-ativo de componentes de software;
- identificar questões de propriedade intelectual que surgem durante o desenvolvimento de software e prover um meio de acompanhar suas resoluções;

- fornecer um ambiente colaborativo em que os times de desenvolvimento e de aconselhamento legal podem acessar a informação necessária para tomada de decisão de maneira eficiente;

- automatizar a conformidade com políticas de propriedade intelectual corporativas, permitindo a implementação de processos de negócio para apoiar as políticas.

Em 2 de fevereiro de 2010, a Black Duck conseguiu a patente US 7.552.093 B2, acerca de sua tecnologia para verificar automaticamente as obrigações das licenças e conflitos. Essa patente causou uma certa controvérsia na comunidade de software livre, que em geral é contra qualquer patente de software. Bradley Kuhn começou a discussão com um artigo em seu site [Kuh10] afirmando que a patente descreve um processo que ele já vinha executando regularmente desde 1999. A invenção patenteada é considerada "óbvia" por ele e outras pessoas que já participavam de tarefas de análise de licenças de software livre, o que poderia tornar a patente inválida se levada ao tribunal. Chris DiBona, gerente de software livre do Google, afirma que o próprio Google possui *prior art* em relação a essa patente [DiB10], o que é suficiente para invalidá-la.

Em decorrência da patente e pelo fato das soluções da Black Duck serem comercializadas como software fechado, a visão de alguns membros da comunidade de software livre é que essa empresa tem como objetivo se aproveitar do receio das pessoas em relação ao software livre, fazendo com que elas comprem ferramentas que elas não necessitam, pois se manter em conformidade com as licenças é mais fácil do que eles fazem parecer [Kuh10].

7.7 Palamida

A Palamida (www.palamida.com) é outra empresa americana, fundada em 2003, especializada no serviço de consultoria e auditoria referentes ao uso de software livre. Assim como a Black Duck, eles têm uma tecnologia patenteada para analisar o código fonte, fazer um levantamento do que ele contém e alertar sobre possíveis problemas. A empresa foi fundada em 2003, após seus fundadores terem uma experiência desfavorável com uso de um software livre que não estava bem documentado no projeto. A partir daí resolveram desenvolver uma ferramenta para auxiliar no processo de identificar, rastrear e gerenciar o conjunto de componentes de um projeto, para não apenas ter conhecimento sobre a propriedade intelectual envolvida, como também estar ciente de possíveis vulnerabilidades de segurança.

Apesar da Palamida ser um membro do *Open Compliance Program* da *Linux Foundation*, que será discutido adiante, ela é bem menos conhecida do que a Black Duck [2]. A falta de informações e a indisponibilidade de uma versão gratuita do produto não facilitam uma análise mais detalhada das soluções oferecidas pela Palamida, porém, isso não a desqualifica como uma empresa a ser considerada caso surja a necessidade de um

[2] Uma pesquisa no Google pelos termos *"black duck" software license* retornou, em 23 de maio de 2011, 186 mil resultados, enquanto uma busca por *palamida software license* retornou apenas 3.690 resultados.

suporte comercial em questões de propriedade intelectual e vulnerabilidades relacionadas a software livre.

7.8 Microformatos

Microformatos são itens de marcação semântica para documentos na Internet, adotados por convenção a partir da adaptação de padrões já em uso, que tem como finalidade facilitar a extração da semântica dos dados tanto por humanos como também por máquinas. As especificações de cada microformato são desenvolvidas pela comunidade, utilizando como ferramenta principal o wiki disponível no site do projeto: microformats.org.

Uma das áreas em estudo pela comunidade é a de licenças [Lin], visando facilitar a forma como autores e publicadores indicam as exigências de atribuição e licenciamento do conteúdo disponibilizado. Desde 2004 é usado o atributo *rel="license"* em tags de *link* HTML para indicar a existência de uma licença para o conteúdo da página [Çel]. Quando há mais de um *link*, o significado atribuido é que qualquer uma das licenças pode ser utilizada. O uso do *rel="license"* é patenteado, porém livre de *royalties*. Várias ferramentas bastante conhecidas utilizam esse formato para identificar a licença em páginas da web, tais como o Dreamweaver, Yahoo! (cc) Search e Google "Usage Rights" Search.

Apesar desse método ser mais utilizado em conteúdo de texto, fotos, vídeo, áudio ou similares, nada impede seu uso também em software. Esta modalidade traria vantagens como o acesso rápido e automatizado à documentação relacionada a licenças diretamente a partir de uma interface web do aplicativo (por exemplo o controle de versão), reduzindo a possibilidade de erro de outras ferramentas já citadas neste capítulo.

7.9 Linux Foundation Open Compliance Program

O *Open Compliance Program* é uma iniciativa da *Linux Foundation*, anunciada em agosto de 2010, para orientar indivíduos e empresas em questões relacionadas a licenciamento de software, com o objetivo de aumentar a adoção de software livre e reduzir o receio existente no mercado. A fundação oferece uma série de artigos e também alguns cursos, com as opções de treinamento em empresas ou com um instrutor remoto, com duração que varia de duas horas a dois dias. Os cursos oferecidos são [Had]:

- *Implementation and Management of Open Source Compliance*;
- *Overview of Open Source Compliance End-to-End Process*;
- *Executive Review of Open Source Compliance*;
- *Open Source Compliance Programs: What You Must Know.*

A *Linux Foundation* também desenvolveu algumas ferramentas que complementam as que já foram estudadas aqui anteriormente. São elas:

- *Dependency Checker*: identifica combinações de código por ligação estática ou dinâmica e provê um arcabouço de política de licenciamento que ajuda a identificar combinações de licença e ligação que devem ser destacadas durante o monitoramento de licenças de um projeto;
- *The Code Janitor*: auxilia os desenvolvedores a limparem do código termos e comentários que devem ser removidos antes do lançamento do software;
- *Bill of Material (BoM) Difference Checker* (ainda em planejamento): identifica alterações no código fonte de componentes incluidos em um projeto.

As ferramentas estão disponíveis, de forma livre, no repositório git da própria *Linux Foundation*, em git.linuxfoundation.org.

Por outro lado, as ferramentas não são suficientes para garantir que as empresas não estejam violando as licenças de software. É muito importante que elas estejam acompanhadas de processos que permitam o acompanhamento do ciclo de vida do projeto. Para auxiliar nessa tarefa, um dos artigos mais importantes publicados pela fundação é o *Self-Assessment Checklist*, que é uma lista de práticas compiladas a partir da experiência de diversas empresas que possuem programas de conformidade para seu software. A lista de práticas abrange as seguintes áreas [The10]:

- Descoberta e revelação: identificação do software de terceiros, incluindo software livre;
- Análise e aprovação: revisão do uso planejado de software livre na distribuição de um produto e, se necessário, em projetos internos;
- Satisfação das obrigações: verificação das obrigações impostas pelas licenças referentes ao software livre utilizado (notas de *copyright* e atribuição, cópia da licença, disponibilização do código fonte, etc.);
- Contribuições à comunidade: revisão das contribuições feitas por funcionários que são incorporadas aos projetos da comunidade.

O checklist ainda inclui elementos de suporte, tais como política da empresa, habilidades necessárias para que funcionários da empresa exerçam as atividades relacionadas ao programa de conformidade, processos de negócio, treinamento e gerenciamento.

Portanto, para garantir que os processos de uma equipe estão alinhados com as exigências das licenças de software livre, além de todas as ferramentas já vistas nesse capítulo, o checklist da *Linux Foundation* possui um papel fundamental. Ele pode ser obtido em www.linuxfoundation.org/programs/legal/compliance/self-assessment-checklist.

No próximo capítulo, estudaremos o impacto do licenciamento em alguns projetos de software livre proeminentes.

Capítulo 8

Estudos de Caso

Muitas licenças de software surgem em resposta a necessidades específicas de determinados projetos, a ponto de receberem o nome do projeto ou da entidade que motivou sua existência (exemplos incluem a *Eclipse Public License* e a *Apache License*). Um estudo sobre licenças não estaria completo sem abordar alguns destes projetos, portanto, neste capítulo, estudaremos aspectos de licenciamento de projetos proeminentes de software livre.

O artigo *Intellectual Property Policy and Attractiveness: A Longitudinal Study of Free and Open Source Software Projects* [SCK+11] mostra de forma estatística o impacto que uma mudança de licença pode causar na atratividade de um projeto. Já o *An Exploratory Study of the Evolution of Software Licensing* [DPGGA10] faz uma análise quantitativa e qualitativa do impacto das mudanças de licenciamento, tomando como base alguns projetos. Embora o impacto deste tipo de mudança já tenha sido abordado no Capítulo 7, as informações levantadas no artigo fundamentam várias das análises apresentadas a seguir.

8.1 MySQL

O armazenamento de dados é parte fundamental de praticamente qualquer projeto de software. A abordagem de delegar essa tarefa a um sistema de bancos de dados relacional ganhou popularidade desde que o modelo relacional foi concebido por E. F. Codd nos anos 70 [Cod70] e diversas empresas (ex.: IBM, Oracle, Sybase, Informix, Microsoft) passaram a vender sistemas de gerenciamento de bancos de dados (SGBDs). Os sistemas mais robustos eram, em sua maioria, fechados, mas a popularização do formato de desenvolvimento aberto fez com que surgissem alternativas, dentre as quais o PostgreSQL e o MySQL ganharam posições de destaque.

No caso do MySQL, o desenvolvimento do software se deu, em sua maior parte, sob a responsabilidade de uma empresa particular, a MySQL AB, cuja receita se baseava, por um lado, no suporte e treinamento que ela dava a clientes corporativos, e, por outro lado, no modelo de licenciamento duplo: o MySQL Server foi distribuído ao longo do tempo através de licenças livres (atualmente a GPL 2.0 com uma exceção, conforme será visto

adiante), porém, clientes que desejam fazer uso fora dos limites desta podem licenciar o software através de contratos comerciais específicos.

Esse modelo garantiu a melhoria contínua do produto, fazendo com que muitos desenvolvedores o adotassem, e, por conta disso, muitos ambientes e linguagens de desenvolvimento passaram a incorporar as bibliotecas de acesso ao MySQL Server - ou "conectores", como são denominadas na documentação. Essas bibliotecas eram licenciadas através da LGPL, o que permitia a sua incorporação através de seus termos (conforme descrito na Seção 5.3.1). No entanto, a empresa estava preocupada com a inclusão destes conectores em software fechado (para o qual esperava um acordo de licenciamento, conforme o modelo comercial descrito acima), e em 2004 alterou a licença desses conectores para a GPL 2.0 [DPGGA10].

Tal alteração causou uma série de efeitos colaterais em iniciativas de software livre. Um dos casos mais contundentes foi o da linguagem PHP: após a mudança, sistemas PHP não podiam mais se conectar com o banco de dados MySQL, pois a licença do PHP (uma licença personalizada, reconhecida como "licença de software livre não-*copyleft*" pela *Free Software Foundation* [Fre]) não é compatível com a GPL (em particular por restringir o uso do termo "PHP" nos nomes de trabalhos derivados). O PHP e o MySQL estão intimamente ligados: a escolha deles como base para o desenvolvimento de software, aliados ao sistema operacional Linux e ao servidor web Apache é tão comum que o conjunto dos quatro pacotes de software (ou pequenas variações equivalentes, como por exemplo NetBSD no lugar de Linux) ganhou seu próprio acrônimo: LAMP (Linux, Apache, MySQL e PHP).

A ruptura entre PHP e MySQL seria desastrosa para o ecossistema do software livre. A solução oferecida pela empresa foi a criação da *MySQL FOSS Exception* [Ora]: uma cláusula extra que permite a distribuição das bibliotecas/conectores junto a software cuja licença esteja em uma lista de licenças autorizadas. A lista contém, claro, a PHP License (3.0 e 3.1), e mais 27 licenças, incluindo todas as que foram vistas no Capítulo 5.

Outro aspecto relevante desse projeto foi o impacto sobre o licenciamento causado pelas sucessivas aquisições de empresas sob as quais seu desenvolvimento se deu, bem como de empresas que possuíam tecnologias relevantes para o projeto. Uma característica do MySQL é a capacidade de usar diferentes formatos de armazenamento para as tabelas, índices e outras estruturas gerenciadas pelo programa. Esta flexibilidade se dá graças ao uso de diferentes motores de armazenamento, e esta abordagem se beneficia em larga escala do modelo de licenciamento de software livre, já que diferentes programadores e empresas podem fornecer e melhorar estes motores. Um exemplo notável da eficácia desta abordagem foi o surgimento do InnoDB, um motor que melhorava sensivelmente o suporte do banco de dados a comportamentos transacionais e de integridade referencial, quando comparado ao MyISAM, que era o motor padrão até então.

O código fonte do InnoDB, inicialmente fechado, passou a ser duplamente licenciado, adotando a licença GPL 2.0 para viabilizar sua distribuição juntamente com o MySQL, através de contrato especial entre a Innobase OY (seu fabricante) e a MySQL AB. No

entanto, em Outubro de 2005 a Oracle comprou a Innobase OY, e a posição da empresa como vendedora de um sistema fechado de banco de dados, cujas vendas poderiam ser impactadas por um software livre como o MySQL, colocava em xeque a renovação do contrato mencionado. No entanto, o mesmo foi renovado, e o desenvolvimento prosseguiu. Porém, o mesmo padrão se repetiu um ano depois: em Fevereiro de 2006 a mesma Oracle compraria a Sleepycat Software, criadora do Berkley DB e do motor BDB. Nesse caso, o impacto era um pouco menor, pois esse motor era pouco utilizado, e optou-se por remover o suporte a partir da versão 5.1 do MySQL.

As questões com licenças e aquisições se acelerariam com a compra da própria MySQL AB pela Sun Microsystems, concretizada em Janeiro de 2008. A princípio, essa aquisição não alterou o estado das coisas: o MySQL continuou disponível no sistema de licenciamento duplo. No entanto, em Abril de 2009, a Sun foi comprada pela Oracle, uma aquisição que a princípio chamou a atenção pelo fato da primeira ser a detentora de uma série de patentes e direitos autorais relacionados ao Java (mesmo tendo licenciado o núcleo das ferramentas ligadas a plataforma sob licenças livres nos anos anteriores), mas que também preocupou os usuários e desenvolvedores que usavam MySQL. Durante as negociações com a Comissão Européia para aprovar a aquisição, a Oracle se comprometeu a dar continuidade ao modelo de licenciamento duplo do MySQL pelo menos até o ano 2015. Porém, pelo menos um *fork* surgiu em consequência da divergência de interesses em relação a essa aquisição: o MariaDB (mariadb.org). Esse é um trabalho desenvolvido de forma independente, e é licenciado exclusivamente sob a GPL 2.0 (incluindo a cláusula de exceção para permitir o empacotamento de conectores, conforme visto acima), e se destaca também pela inclusão do XTraDB (www.percona.com/software/percona-xtradb), um substituto com 100% de compatibilidade reversa com o InnoDB, distribuído exclusivamente sob a GPL 2.0.

8.2 Eclipse

O Eclipse é um arcabouço para a criação de ambientes integrados de desenvolvimento de software (IDEs) com suporte para múltiplas linguagens de programação, configurações e plataformas alvo, através de *plug-ins* que muitas vezes são desenvolvidos de forma independente do projeto principal. Essa diversidade de públicos deu origem a diversas distribuições (muitas vezes sujeitas a *branding* totalmente diferenciado), mas o software base é distribuído sob a *Eclipse Public License* (discutida na Seção 5.3.3).

Cada distribuição alternativa acaba tendo que lidar com a questão do sublicenciamento de acordo com suas próprias necessidades. Um exemplo é o Aptana Studio, uma IDE que remodela o Eclipse tornando-o atrativo para a criação de aplicativos web dinâmicos (AJAX). O Aptana Studio era distribuído sob um modelo de licenciamento duplo até a versão 2, que permitia ao usuário optar entre uma versão da GPLv3 com uma exceção específica para as dependências oriundas do Eclipse ou uma licença própria, a *Aptana Public License* (APL). A partir da versão 3, o Aptana está disponível a princípio

apenas sob a GPL, com uma exceção chamada *Aptana-GPL exception*, porém permite que organizações façam uso interno do produto sob a APL e está aberta a propostas comerciais para definir outros termos de licença através de um acordo entre as partes. A escolha da GPL faz com que não seja possível redistribuir o Aptana com outros *plugins* do Eclipse que estão sob a EPL [Apt11].

Outro aspecto notável é que o desenvolvimento do Eclipse deu origem ao SWT, uma biblioteca de componentes gráficos Java alternativa àquelas criadas pela Sun (AWT e Swing), que une o visual nativo oferecido pela AWT com a compatibilidade multiplataforma encontrada na biblioteca Swing, sem grande sobrecarga no desempenho. Essa flexibilidade tornou-a atrativa para projetos não relacionados ao Eclipse ou a IDEs em geral, e a menor incidência de elementos *copyleft* na EPL não desestimulou o licenciamento livre de muitos programas que a utilizam. Exemplos incluem diversos projetos: alguns livres como o cliente do projeto Haystack do MIT (groups.csail.mit.edu/haystack) e o Sistema de Informação Geográfica (GIS) uDig (udig.refractions.net); sistemas fechados como o Lotus Notes (www-01.ibm.com/software/lotus/products/notes); e até software com licenciamento duplo, como o sistema de compartilhamento de arquivos Vuze/Azureus (www.vuze.com).

8.3 Mono

Um dos fatores chave de sucesso do sistema operacional Windows é a ampla disponibilidade de aplicativos. Esta, por sua vez, está relacionada com a variedade de ferramentas de desenvolvimento disponíveis para ele. Além de contar praticamente com todas as ferramentas disponíveis para sistemas operacionais baseados em UNIX (ex.: gcc, make, Eclipse), a plataforma tem o histórico de ter sido o berço dos ambientes de desenvolvimento RAD (*Rapid Application Development*), e o próprio fabricante do sistema operacional (Microsoft) foi responsável pela criação de um dos ambientes mais populares: a plataforma .NET.

A plataforma Java já chamava a atenção da Microsoft desde os anos 90, pois sua característica de permitir a execução de software independente do sistema operacional utilizado (desde que ele possa executar uma máquina virtual Java) transformava o principal produto da empresa em *commodity*. Após uma tentativa mal sucedida de subverter a plataforma através da prática de "abraçar e estender" o padrão, isto é, adotando-o e introduzindo incompatibilidades que amarram o desenvolvedor à sua versão particular da linguagem, a Microsoft decidiu tomá-lo como exemplo, trabalhando uma nova plataforma sem compromisso de compatibilidade retroativa, e essa plataforma, anunciada em Junho de 2000, viria a se tornar o .NET.

O aspecto mais surpreendente desse esforço foi que as especificações dos componentes centrais da plataforma, isto é, da linguagem C#, da CLI (*Common Language Infrastructure*, que inclui tanto a máquina virtual quanto a biblioteca padrão de classes) e da conexão C++/CLI são todas de padrão aberto (foram submetidas ao ECMA e ao ISO),

permitindo que qualquer pessoa ou empresa implemente sua própria versão dos compiladores e classes necessários para compilar e executar aplicativos para esta plataforma, sem a necessidade de obter autorização da Microsoft. Padrões abertos são muito importantes e caminham lado a lado com o software livre O próprio Bruce Perens, que escreveu a definição de código aberto (vide Seção 2.2.1), também criou em seu site uma página intitulada *Open Standards, Principles and Practice* em que propõe uma definição com as características dos padrões abertos [Bru10]. Lawrence Rosen também dedica um capítulo inteiro de seu livro sobre licenças de software livre [Ros05] para o tema de padrões abertos.

Miguel de Icaza iniciou o projeto GNOME com Federico Mena em 1997, sendo um de seus principais objetivos trazer a maior quantidade possível de software para o Linux, o que resultou na fundação da empresa Helix Code, em 1999. A empresa foi renomeada para Ximian e adquirida pela Novell, e sob seus auspícios foi produzida a primeira versão do Mono, uma implementação livre do compilador C# e da CLI que permite o desenvolvimento e execução cruzados entre Windows, Linux e outras plataformas (que vão desde celulares com Android e iOS até consoles de videogame como Playstation, Wii e XBox 360). O anúncio do projeto foi feito em 2001, e uma versão 1.0 foi lançada em Junho de 2004.

Inicialmente, o projeto era distribuído sob a GPL 2.0 (com as bibliotecas de tempo de execução distribuídas sob LGPL 2.0). Porém, conforme discutido no Capítulo 6, há uma certa insegurança em relação à liberdade da escolha da licença quando utiliza-se APIs ou uma máquina virtual GPL, pois alguns consideram que essa situação implicaria em um trabalho derivado, que portanto também deveria ser GPL. Por esse motivo, os desenvolvedores do Mono optaram por alterar a licença das bibliotecas de classe e adotar a MIT (que já era utilizada na biblioteca de classes em tempo de compilação), apresentada no Capítulo 5, que é bem mais simples e permissiva, permitindo claramente seu uso e distribuição junto a sistemas que usam outra licença, tanto de software livre como software fechado [DPGGA10].

Tal alteração foi um fator decisivo para que outras empresas, como a HP, também contribuíssem para o projeto. De forma análoga, o compilador C# era originalmente licenciado sob a GPL 2.0 e posteriormente adotou o modelo de licenciamento duplo (vide Seção 5.2.1) para incluir a MIT. Essa mudança permitiu também o reuso de código do compilador dentro de outros componentes [Mon], reafirmando as vantagens de uma licença permissiva em um projeto dessa natureza.

Outro aspecto notável acerca da flexibilidade que o licenciamento adotado permitiu é a resiliência que o projeto demonstrou diante das mudanças ocorridas nas empresas sob as quais ocorreu seu desenvolvimento. Quando a Ximian foi adquirida pela Novell (2003), esta anunciou seu suporte à continuidade do Mono e projetos relacionados, mas uma nova aquisição em Março de 2010 (dessa vez a Novell seria adquirida pela Attachmate) levaria a uma série de demissões e cancelamentos, fazendo com que o próprio Icaza optasse por criar uma nova empresa, a Xamarin (www.xamarin.com), que iria se dedicar ao

desenvolvimento futuro da plataforma. É importante observar que todas essas mudanças ocorreram sem prejuízo para desenvolvedores, colaboradores ou usuários do Mono.

8.4 A biblioteca Qt no contexto do KDE

A biblioteca Qt é bastante usada no desenvolvimento da interface gráfica de aplicativos para *desktop* e celulares. A Qt foi desenvolvida pela empresa Trolltech e, desde 2008, pertence à Nokia (qt.nokia.com).

Inicialmente, a biblioteca não era software livre, e era disponibilizada por uma licença gratuita, a *FreeQt License*, e outra comercial. Em 1998, foi firmado um acordo entre a Trolltech e o KDE, garantindo a continuidade da licença gratuita e atualização periódica da biblioteca, caso contrário a Qt seria relicenciada como BSD.

Apesar do acordo, havia um clima de resistência e incerteza na comunidade de software livre, devido à falta de garantias de que a biblioteca continuasse evoluindo de uma forma condizente com as necessidades do projeto KDE. Por esse motivo, eventualmente a Trolltech optou por criar uma licença que seguia os princípios do software livre, a *Q Public License)*. Essa nova licença foi aprovada pela *Open Source Initiative*, porém ainda considerada incompatível com a GPL pela *Free Software Foundation*.

Como a maior parte dos aplicativos do KDE eram GPL, ao seguir essa interpretação, havia uma violação da licença [Swe]. O Projeto KDE discordava da incompatibilidade e defendia a ideia de que não havia nenhum problema legal do uso da QPL com a GPL, porém a desconfiança da comunidade em relação a essa questão fortaleceu o projeto Gnome, que utilizava principalmente a biblioteca GTK ao invés da Qt, e também deu origem ao projeto Harmony, cuja ideia era implementar um substituto da Qt licenciado sob a GPL [DPGGA10]. A forma encontrada para resolver a questão e tentar recuperar a força da comunidade de desenvolvimento da Qt foi relicenciar a biblioteca sob a GPL, sendo que atualmente há algumas alternativas de licença disponíveis:

- LGPL 2.1 [Noka]
- GPLv3 com uma exceção que permite que o aplicativo final seja distribuído sob algumas licenças de software livre incompatíveis com a GPL [Nokb]
- *Qt Commercial Developer License* [Nokc]

Com a alteração da licença, o projeto Harmony foi abandonado [DPGGA10].

8.5 Núcleo Linux

Ao lado do Apache, o Linux foi, sem sombra de dúvida, um dos projetos mais importantes no sentido de popularizar o conceito e a prática de uso das licenças de software livre. Sua criação foi providencial para fornecer ao Projeto GNU (cujo histórico foi detalhado na Seção 2.2) um núcleo amplamente adotado, componente indispensável para

que o mesmo pudesse compor um sistema operacional completo para o usuário final. No entanto, esta combinação enfrentava um problema de compatibilidade de licenças logo na sua concepção. A documentação que acompanhava a primeira versão pública (0.01) declarava que [Tor91]:

```
This kernel is (C) 1991 Linus Torvalds, but all or part of it may be
redistributed provided you do the following:

- Full source must be available (and free), if not with the distribution
  then at least on asking for it.

- Copyright notices must be intact. (In fact, if you distribute
  only parts of it you may have to add copyrights, as there aren't
  (C)'s in all files.) Small partial excerpts may be copied without
  bothering with copyrights.

- You may not distibute this for a fee, not even "handling" costs.
```

Essa licença restringia várias possibilidades de uso comercial, tornando-a ainda mais restritiva que a própria GPL. Imaginar o uso do Linux sem o conjunto de software GNU não era uma possibilidade, a própria distribuição incluía uma cópia binária do *shell* Bash do projeto GNU (sem a qual não seria possível interagir com o núcleo ou iniciar programas a partir dele), e compiladores e outras ferramentas GNU seriam indispensáveis para o seu uso (tanto quanto o Linux tornou-se indispensável para o projeto GNU, já que o desenvolvimento do núcleo GNU Hurd caminhava a passos lentos).

A afinidade dos projetos e a popularidade que o núcleo Linux atraiu em um curto espaço de tempo motivou a resolução do impasse: a partir da versão 0.99, o núcleo Linux passava a ser licenciado pela GPL, formando um conjunto cuja sinergia é tamanha que torna comum a confusão entre o núcleo Linux e o sistema operacional GNU/Linux [Sta97]. Mas a mudança ainda não resolveria todas as questões de compatibilidade de licenças relacionadas ao núcleo Linux.

Um aspecto que ainda gera controvérsia, por exemplo, é a questão em torno dos *Loadable Kernel Modules* serem considerados trabalhos derivados ou não. O assunto é relevante devido ao fato de alguns fabricantes de hardware se recusarem a disponibilizar informações que permitam a criação de controladores de dispositivo livres, forçando os mantenedores do núcleo a lançar mão da incorporação de trechos binários de código fechado disponibilizado pelos próprios fabricantes. Se tais controladores de dispositivo forem considerados trabalhos derivados, a distribuição do Linux viola a própria GPL sob a qual ele é licenciado [Sta02].

Sob a luz dos critérios delineados na Seção 6.1, podemos considerá-los como bibliotecas de ligação dinâmica (e, portanto, descaracterizá-los como trabalhos derivados das interfaces utilizadas), mas esta modalidade de ligação carece de jurisprudência específica, mantendo a questão em aberto. Um efeito colateral deste tipo de "zona cinza" legal é o surgi-

mento de projetos como o Linux-libre (www.fsfla.org/svnwiki/selibre/linux-libre/index), que mantém versões modificadas do núcleo Linux, removendo as partes que não contém código fonte, ou que contém código fonte ofuscado ou licenciado sob licenças não livres.

Outra questão delicada envolvendo licenças e o núcleo Linux está relacionada ao uso da GPL 2.0 no lugar da GPLv3. A Seção 5.2.2 já trata destas diferenças, mas é importante observar como elas se manifestaram no núcleo Linux. O núcleo é licenciado através da GPL 2.0 e não contém, em sua licença atual, a tradicional opção de usar qualquer versão posterior que o usuário deseje. Isso acontece em grande parte porque o próprio Torvalds manifesta resistência ao uso da GPLv3, e apenas o código contribuído por ele já seria um grande obstáculo em termos de conversão. Sua posição inicial foi de que [Tor06]:

> *"The Linux kernel is under the GPL version 2. Not anything else. Some individual files are licenceable under v3, but not the kernel in general.*
>
> *And quite frankly, I don't see that changing. I think it's insane to require people to make their private signing keys available, for example. I wouldn't do it. So I don't think the GPL v3 conversion is going to happen for the kernel, since I personally don't want to convert any of my code."*

Mais adiante, ele conclui:

> *"Conversion isn't going to happen."*

Esta resistência está relacionada justamente às salvaguardas extras contra fenômenos como a *tivoização*, discutidas na Seção 5.2.2, e é relevante observar que nem todos os desenvolvedores veem elas como vantagem, haja visto que elas podem limitar a liberdade de uso do código em determinadas situações. A polêmica deu origem a diversos estudos, sendo que Laffan [Laf07] usa o repositório de código fonte SourceForge como base para afirmar que, em Setembro de 2007, 500 dos 6000 projetos GPL 2.0 hospedados no site já haviam sido convertidos para GPLv3, isso com apenas dois meses de publicação da nova licença. Este número certamente aumentou com os anos, mas até a data de publicação desta dissertação o núcleo Linux permanece na GPL 2.0, sem perspectivas de mudança.

8.6 Android

É razoável comparar o impacto do sistema operacional Android no universo dos dispositivos móveis àquele causado pelo Apache no mercado de servidores web, ou mesmo pelo Linux no tocante a sistemas operacionais. O sistema surgiu da empresa homônima adquirida pelo Google em 2005, mas é oficialmente mantido pela *Open Handset Alliance*, embora o Google ainda supervisione o desenvolvimento e seja o maior contribuidor de código. Esse sistema merece observação detalhada não apenas por reunir diversos componentes de software com licenças nem sempre compatíveis, mas também por se apresentar como uma alternativa livre em um meio tradicionalmente dominado por software fechado.

O estudo *Android - Opportunity, Complexity, and Abundance* [Blaa] informa que o sistema é composto de cerca de 185 componentes distintos, abrangendo em torno de 19 licenças de software livre, a maior parte delas recíprocas, e algumas sequer aprovadas pela OSI. No geral, o código contribuído pelo próprio Google usa a licença Apache 2.0: a empresa justifica [Goo] a escolha desta licença sobre outras licenças com características de *copyleft* (em particular a LGPL) alegando que esta limitaria a liberdade dos fabricantes e integradores acerca de como poderiam incluir o software em seus dispositivos, isto é:

> *"Android is about freedom and choice. The purpose of Android is promote openness in the mobile world, but we don't believe it's possible to predict or dictate all the uses to which people will want to put our software. So, while we encourage everyone to make devices that are open and modifiable, we don't believe it is our place to force them to do so. Using LGPL libraries would often force them to do so."*

Uma preocupação específica é o fato de a LGPL requerer que o usuário tenha a capacidade de substituir bibliotecas ligadas ao sistema por outras versões, o que seria um problema quando o software de sistema é distribuído em imagens binárias estáticas, muitas vezes em midias *flash* somente de leitura (um formato tradicional para celulares e dispositivos embarcados). Além disso, alguns fabricantes se recusariam a permitir este tipo de liberdade e, ao mesmo tempo, oferecer garantias de funcionamento. Finalmente, a empresa manifesta preocupação com problemas históricos de aderência à LGPL.

A presença de código de terceiros, no entanto, leva à necessidade de flexibilizar esta estratégia. O ponto mais evidente é o núcleo: sob a interface gráfica do sistema Android, opera um núcleo Linux, licenciado sob GPL 2.0. Isso significa que quaisquer *patches* feitos no sistema precisam ser licenciados sob a GPL 2.0. É importante notar que estas contribuições não são poucas, dada a natureza de um projeto que envolve dezenas de fabricantes diferentes, cada um com diversos modelos de hardware.

A isso se adiciona o fato de que os fabricantes também fazem suas próprias modificações. Algumas são puramente cosméticas e podem até ser implementadas em nível de aplicação, mas outras requerem modificações diretas no sistema em si. Estas modificações não apenas precisam obedecer aos termos das licenças já existentes, mas também precisam verificar a compatibilidade de qualquer software novo introduzido nesse momento com o ecossistema descrito acima.

Uma outra estatística interessante levantada pelo estudo é que as licenças dos diferentes componentes totalizam cerca de 1700 obrigações, isto é, itens que devem ser verificados ou cumpridos pelos implementadores para que não haja qualquer violação dessas licenças. Indo além, ele separa elas em cerca de 1000 obrigações legais e aproximadamente 700 obrigações específicas do desenvolvimento de software. Apesar de notar que as obrigações legais podem ser resolvidas em uma revisão jurídica única, as outras demandam gerenciamento, que muitas vezes deve ser feito arquivo a arquivo no software. O uso de ferramentas e metodologias como as mencionadas no Capítulo 7 torna-se indispensável

para qualquer um interessado em adaptar o sistema para uso em hardware específico.

Mas a questão mais acalorada em torno das licenças de software do Android está em uma camada ligeiramente acima do núcleo: a máquina virtual Dalvik. Os sistemas operacionais para dispositivos móveis mais avançados (*palmtops*, *smartphones*, *tablets* etc.) normalmente seguem dois possíveis caminhos para a instalação de aplicativos de terceiros: ou estes aplicativos são distribuídos como arquivos binários que rodam diretamente, que é o caminho adotado por sistemas como Windows Mobile, webOS (Palm OS) e iOS, ou estão no formato de *bytecode* voltado a uma máquina virtual oferecida pelo celular, abordagem oferecida por exemplo pelo BlackBerry OS. Alguns sistemas, como o Symbian, oferecem ambas as possibilidades, e quase sempre a máquina virtual para celulares em questão é uma máquina virtual Java e os aplicativos obedecem a algum conjunto dos perfis/especificações definidos pela Sun sob o conjunto de especificações coletivamente denominado *Java Platform, Micro Edition* (Java ME). MIDP 2.0 é um perfil típico em aparelhos deste tipo, mas cada fabricante implementa/licencia o subconjunto que julga mais apropriado.

O Android adotou Java como linguagem para o desenvolvimento de aplicativos, mas a inclusão de um padrão fechado seria incompatível com o seu modelo. Isso, aliado às limitações técnicas de uma especificação tão antiga (sua primeira versão é de 1999, quando *smartphones* ainda sequer eram um conceito) e histórico de incompatibilidade entre implementações Java ME levou os projetistas do Android a outro caminho: criaram o Dalvik, uma nova máquina virtual otimizada para a realidade do Android (entre outras coisas, ela é projetada para que várias máquinas virtuais estejam em execução ao mesmo tempo).

Um aspecto bastante questionado acerca do uso de Java em ambientes de software livre é que mesmo com o processo relativamente aberto de especificações (JSRs) e com a publicação de diversos componentes como software livre, a Oracle ainda detém bastante controle sobre diversos aspectos da plataforma, incluindo a proteção bastante zelosa da marca Java, impedindo o seu uso em implementações que não sejam absolutamente compatíveis, e seguramente não aprovaria a criação de um padrão diferenciado (e não licenciado) para dispositivos móveis. O Dalvik permitiu ao Android superar esta dificuldade, pois não executa *bytecode* Java nem usa as bibliotecas de classes Sun.

Os desenvolvedores de aplicativos usam um conjunto semelhante, porém distinto, de classes, mais especificamente um subconjunto do Apache Harmony, cuja licença é, como se esperaria, a mesma licença Apache do Android. Eles também não distribuem classes Java: embora usem compiladores Java durante o processo de desenvolvimento dos aplicativos, um utilitário fornecido pela *Open Handset Alliance* no SDK do Android converte o *bytecode* Java (*.class*) para *bytecode* Dalvik (*.dex*). Isso faz com que o Dalvik não seja, legalmente, uma "máquina virtual Java", e, portanto, evita que ele esteja sujeito a licenciamento por parte da Oracle.

No entanto, essa estratégia não foi suficiente para impedir o surgimento de complicações jurídicas. A *Oracle Corporation* adquiriu a *Sun Microsystems* em Abril de 2009 (nomeando-a *Oracle America, Inc*), e, com isso, toda a sua propriedade intelectual. E

em Agosto de 2010 a Oracle processou o Google [Ora10] não pela violação de licenças de software (que, em tese, não ocorreriam, considerando o uso conforme descrito), e sim por violação de patentes e direitos autorais. No momento da publicação desta dissertação, o processo está se desenrolando e as patentes ainda estão sendo analisadas. Qualquer que seja o desfecho, entretanto, o caso ilustra aspectos legais e de gestão de risco na delicada relação entre software livre e software fechado - um risco que Stallman [Sta04], não por acaso, denomina “Armadilha Java”.

Capítulo 9

Conclusão

Espera-se que a metodologia proposta no final do Capítulo 5 sirva como base para orientar o licenciamento de novos projetos de software livre, e que o estudo de compatibilidade entre licenças do Capítulo 6, baseado na categorização que o precedeu, ajude os desenvolvedores a integrarem esse tipo de software em seus próprios projetos, respeitando os termos de cada licença. Esse estudo de compatibilidade evidencia o nível de complexidade que a interação entre as diferentes licenças dos subprojetos, bibliotecas, arcabouços e ferramentas utilizados num mesmo projeto pode alcançar. Negligenciar a gestão dos riscos oriundos desta complexidade não é uma opção, em particular quando essas licenças são colocadas à luz dos aspectos jurídicos que as fundamentam, isto é, nos termos descritos no Capítulo 4.

Os casos estudados no Capítulo 8 ilustram diversas nuances de tudo o que foi estudado, introduzindo um aspecto extra: o dinamismo. Os detentores dos direitos do software podem alterar os termos das licenças do projeto ao longo do tempo, e essas mudanças afetam não apenas seus usuários, mas também outros sistemas diretamente ou indiretamente ligados ao projeto original. Em alguns casos, as mudanças podem limitar ou impossibilitar o reuso em situações específicas, como no caso dos conectores MySQL. Mas o contrário, isto é, a promoção de novos canais de distribuição e reutilização do software por consequência de mudanças no seu licenciamento também é possível: tome-se como exemplo a conversão do Java para o modelo de licenciamento de software livre.

É particularmente notável que, quando essas mudanças ocorrem no sentido de adotar licenças mais permissivas, isso tem impacto direto no aumento do tamanho da comunidade que contribui para o projeto. O impacto é ainda maior quando as licenças permitem a criação de trabalhos derivados em caráter comercial, como no caso do Mono, e podem resultar até no abandono de projetos concorrentes, haja visto o ocorrido com a biblioteca Qt. Mas a mudança de licença pode ter consequências imprevisíveis, e isso se observa no estudo do MySQL, trazendo novamente à tona o aspecto da complexidade.

Neste sentido, as ferramentas e metodologias analisadas no Capítulo 7 merecem observação aprofundada, já que elas apresentam o potencial para limitar esta complexidade. No entanto, a análise mostra que elas ainda não estão suficientemente amadurecidas a ponto de resolver diretamente a questão, e novos estudos far-se-ão necessários para que

elas possam, no futuro, garantir a conformidade de um projeto com todas as licenças de seus componentes. Até que esse dia chegue, os usuários, gestores, desenvolvedores e quaisquer outras pessoas relacionadas com o processo de desenvolvimento de software deverão compreender os aspectos estudados neste trabalho, e, conforme o caso, aprofundar-se nos tópicos relevantes. Esse processo de aprendizado não se restringe àqueles que trabalham com software livre, já que, como observado no Capítulo 3, a disseminação do software livre no mercado faz com que seja cada vez menos provável que um projeto de Tecnologia da Informação (seja ele de desenvolvimento ou apenas envolvendo o uso do software) seja desenvolvido e executado sem que haja o envolvimento de algum software livre no processo.

Referências Bibliográficas

[Apt11] Aptana. Legal FAQ. http://www.aptana.com/legal. Acesso em: 12/08/2011, 2011. 90

[ASA10] Thomas A. Alspaugh, Walt Scacchi e Hazeline U. Asuncion. Software Licenses in Context: The Challenge of Heterogeneously-Licensed Systems. *Journal of the Association for Information Systems*, 11:730–755, Nov 2010. 2

[Bar03] Denis Borges Barbosa. *Propriedade Intelectual: direitos autorais, direitos conexos e software*. Lumen Juris, Rio de Janeiro, 2003. 16

[Ben06] Yochai Benkler. *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. Yale University Press, New Haven, 2006. 19

[Blaa] Black Duck Software. Android - Opportunity, Complexity, and Abundance. http://www.blackducksoftware.com/services/androidfaststart. Acesso em: 02/06/2011. 95

[Blab] Black Duck Software. Black Duck Software. http://www.blackducksoftware.com/. Acesso em: 20/02/2011. 82

[Boe10] Eric Boehm. Dispelling Common Legal Myths About Open Source Software Licensing - But why they contain some truth, too. *Ontario Bar Association*, 12(1), Setembro 2010. 72

[Bot06] James Bottomley. GPLv3 Position Statement. http://lkml.org/lkml/2006/9/22/217. Acesso em: 09/01/2011, 2006. 70

[Bru10] Bruce Perens. Open Standards, Principles and Practice. http://perens.com/OpenStandards/Definition.html. Acesso em: 12/08/2011, 2010. 91

[Çel] Tantek Çelik. Microformats Wiki: rel="license" Specification. http://microformats.org/wiki/rel-license. Acesso em: 29/05/2011. 84

[CK08] Martin Campbell-Kelly. Historical Reflections: Will the future of software be open source? *Communications of the ACM*, 51(10):21–23, 2008. 5, 14

[Cod70] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, 13(6):377–387, 1970. 87

[Con98] Congresso Nacional do Brasil. Lei 9.610 - Direitos Autorais. http://www.planalto.gov.br/ccivil/leis/L9610.htm. Acesso em: 06/01/2011, 1998. 67

[Cor] Cornell University Law School. MAI Systems Corp. v. Peak Computer, Inc. http://www.law.cornell.edu/copyright/cases/991_F2d_511.htm. Acesso em: 01/02/2010. 63

[Cor95] W. R. Cornish. Computer Program Copyright and the Berne Convention. *A Handbook of European Software Law*, 1995. 16

[dAMJ05] Juliano Souza de Albuquerque Maranhão e Tercio Sampaio Ferraz Junior. Software Livre: a Administração Pública e a comunhão do conhecimento informático. *Revista de Direito Público da Economia*, 11, 2005. 17

[Det06] Lothar Determann. Dangerous Liaisons - Software Combinations as Derivative Works? *Berkeley Technology Law Journal*, 21(4), 2006. 3, 17, 59, 60, 61, 66, 68, 69, 70

[DiB10] Chris DiBona. License Discuss Mailing List: New patent on license conflicts detection process. http://www.crynwr.com/cgi-bin/ezmlm-cgi?3:mss:16537:201002:dfhnhjjmionjjbgehmap. Acesso em: 08/05/2011, 2010. 83

[dOA97] José de Oliveira Ascensão. *Direito Autoral*. Renovar, 1997. 20

[DOS99] Chris DiBona, Sam Ockman e Mark Stone, editors. *Open Sources*. O'Reilly, Sebastopol, 1999. 7, 8, 13

[DPGGA10] Massimiliano Di Penta, Daniel M. German, Yann-Gaël Guéhéneuc e Giuliano Antoniol. An Exploratory Study of the Evolution of Software Licensing. Em *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, páginas 145–154, New York, NY, USA, 2010. ACM. 71, 75, 87, 88, 91, 92

[dS88] Manoel Joaquim Pereira dos Santos. Software and Copyright: A Marriage of Inconvenience. *The Copyright Magazine*, Junho 1988. 18

[dS08] Manoel Joaquim Pereira dos Santos. *A Proteção Autoral de Programas de Computador*. Lumen Juris, Rio de Janeiro, 2008. 15, 16, 18, 19, 20

[dSP09] Caio Mário da Silva Pereira. *Instituições do Direito Civil*, volume III. Editora Forense, 2009. 21

[ES08] Margaret S. Elliott e Walt Scacchi. Mobilization of Software Developers: The Free Software Movement. *Information Technology & People*, 21(1):4–33, 2008. 1

[FGV] FGV. Centro de Tecnologia e Sociedade - FGV. http://direitorio.fgv.br/cts/licencas. Acesso em: 23/05/2010. 27

[FJL+05] Joaquim Falcão, Tercio Sampaio Ferraz Junior, Ronaldo Lemos, Juliano Maranhão, Carlos Affonso Pereira de Sousa e Eduardo Senna. Estudo sobre o Software Livre. Relatório técnico, Escola de Direito da Fundação Getúlio Vargas, Rio de Janeiro, Março 2005. 17, 19, 21, 22

[For08] Forrester. Open Source Paves the Way for the Next Generation of Enterprise IT, Nov 2008. 13

[Fre] Free Software Foundation. Various Licenses and Comments about Them. http://www.gnu.org/licenses/license-list.html. Acesso em: 06/02/2011. 72, 88

[Fre91] Free Software Foundation. GNU General Public License, version 2. http://www.gnu.org/licenses/gpl-2.0.html. Acesso em: 04/04/2009, 1991. 34, 36

[Fre08] Free Software Foundation. The GNU Operating System. http://www.gnu.org. Acesso em: 17/11/2008, 2008. 45

[Fri88] Mark Friedman. Copyrighting machine language computer software - the case against, 1988. 20

[GH09] D. M. German e A. E. Hassan. License Integration Patterns: Addressing license mismatches in component-based development. Em *IEEE 31st International Conference on Software Engineering*, páginas 188–198, 2009. 3, 71

[Gho05] Rishab Aiyer Ghosh. *CODE: Collaborative Ownership and the Digital Economy*. The MIT Press, 2005.

[GNU] GNU Classpath License. GNU Project. http://www.gnu.org/software/classpath/license.html. Acesso em: 06/02/2011. 71

[Gob] Robert Gobeille. The FOSSology Project - Overview and Discussion. http://www.linuxfoundation.org. Acesso em: 23/04/2011. 76

[Gob08] Robert Gobeille. The FOSSology Project. Em *Proceedings of the 2008 international working conference on Mining software repositories*, MSR '08, páginas 47–50, New York, NY, USA, 2008. ACM. 75, 76, 77

[Goo] Google Inc. Android Open Source Project - Licenses. http://source.android.com/source/licenses.html. Acesso em: 23/05/2011. 95

[Gor] Thomas F. Gordon. Carneades. http://carneades.berlios.de/. Acesso em: 20/02/2011. 79

[Gor10] Thomas F. Gordon. Report on a Prototype Decision Support System for OSS License Compatibility Issues. *Qualipso*, 2010. 79, 80

[Had] Ibrahim Haddad. Linux Foundation Open Compliance Program. http://www.linuxfoundation.org/programs/legal/compliance/training-and-education. Acesso em: 23/02/2011. 84

[Hec78] Paul Heckel. A Technique for Isolating Differences Between Files. *Commun. ACM*, 21:264–268, April 1978. 81

[Hew] Hewlett-Packard Development Company. FOSSology. http://www.fossology.org/. Acesso em: 20/02/2011. 76, 77

[Int09] International Free and Open Source Software Law Review. International Free and Open Source Software Law Review. http://www.ifosslr.org/ifosslr. Acesso em: 12/08/2011, 2009. 2

[KLMS11] Fabio Kon, Nelson Lago, Paulo Meirelles e Vanessa Sabino. Software livre e propriedade intelectual: Aspectos jurídicos, licenças e modelos de negócio. Em SBC, editor, *Jornada de Atualização em Informática (JAI'2011) do Congresso da Sociedade Brasileira de Computação*, páginas 59–107. Editora PUC-Rio, 2011. 3

[Kon01] Fabio Kon. O Software Aberto e a Questão Social. Relatório técnico, Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo, 2001.

[Kuh10] Bradley Kuhn. I Think I Just Got Patented. http://www.ebb.org/bkuhn/blog/2010/02/02/took-our-jobs.html. Acesso em: 08/05/2011, 2010. 83

[Laf07] Liz Laffan. GPLv2 vs GPLv3 - The two seminal open source licenses, their roots, consequences and repercussions. Relatório técnico, VisionMobile, 2007. 38, 94

[Lau04] Andrew M. St. Laurent. *Understanding Open Source & Free Software Licensing*. O'Reilly, Sebastopol, 2004. 26, 27

[Lin] Mike Linksvayer. Microformats Wiki: Licensing. http://microformats.org/wiki/licensing. Acesso em: 20/02/2011. 84

[Mar] Daniel Martins. Licensator. http://licensator.appspot.com/. Acesso em: 20/02/2011. 77

[Mar10] Augusto Tavares Rosa Marcacini. Processo e Tecnologia. Relatório técnico, Faculdade de Direito da Universidade de São Paulo, São Paulo, 2010. 17, 18

[MdC04] Augusto Tavares Rosa Marcacini e Marcos da Costa. *Direito em Bits*. Fiuza Editores, 2004. 15, 21, 22

[Mon] Mono Project, Novell. Mono C# Compiler Under MIT X11 License. http://www.mono-project.com/news/archive/2008/Apr-08.html. Acesso em: 12/05/2011. 91

[Noka] Nokia. LGPL License Option Added to QT. http://qt.nokia.com/about/news/lgpl-license-option-added-to-qt. Acesso em: 23/02/2011. 92

[Nokb] Nokia. Qt GPL Exception Version 1.3. http://doc.trolltech.com/4.4/license-gpl-exceptions.html. Acesso em: 23/02/2011. 92

[Nokc] Nokia. QT Licensing. http://qt.nokia.com/products/licensing. Acesso em: 23/02/2011. 92

[Och04] Tyler T. Ochoa. Copyright, Derivative Works and Fixation: is Galoob a Mirage, or Does the Form(Gen) of the Alleged Derivative Work Matter? *Santa Clara Computer & High Technology Law Journal*, 20(4), 2004. 62

[Opea] Open Source Observatory and Repository. OSCOR - FAQs related to legal questions. http://www.osor.eu/legal-questions-1/faqs-related-to-legal-issues. Acesso em: 06/02/2011. 67

[Opeb] Open Source Observatory and Repository. OSOR - License Compatibility and Interoperability. http://www.osor.eu/legal-questions-1/licence-compatibility-and-interoperability. Acesso em: 06/02/2011. 59, 72

[Ope08] Open Source Initiative. Open Source Initiative. http://www.opensource.org. Acesso em: 22/07/2008, 2008.

[Ora] Oracle. MySQL FOSS License Exception. http://www.mysql.com/about/legal/licensing/foss-exception/. Acesso em: 23/02/2011. 88

[Ora10] Oracle America, Inc. Complaint for Patent and Copyright Infringement and Demand for Jury Trial. http://pt.scribd.com/doc/35811761/Oracle-s-complaint-against-Google-for-Java-patent-infringement?autodown=pdf. Acesso em: 29/04/2011, 2010. 97

[OSO] OSOR. OSOR.EU License Wizard. http://www.osor.eu/legal-questions-1/licence-wizard. Acesso em: 20/02/2011. 78, 79

[OW2] OW2 Consortium. Open Source License Checker. https://wiki.ow2.org/oslcv3/. Acesso em: 20/02/2011. 81

[Ray01] Eric S. Raymond. *The Cathedral & The Bazaar*. O'Reilly, 2001. 11, 75

[Ros05] Lawrence Rosen. *Open Source Licensing: Software Freedom and Intellectual Property Law*. Prentice Hall, New Jersey, 2005. 26, 34, 37, 45, 49, 91

[SCK+11] Carlos Denner Santos, Jr., Marcos Bonci Cavalca, Fabio Kon, Julio Singer, Victor Ritter, Damaris Regina e Tamy Tsujimoto. Intellectual property policy and attractiveness: a longitudinal study of free and open source software projects. Em *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, CSCW '11, páginas 705–708, New York, NY, USA, 2011. ACM. 87

[Sou08] SourceForge. SourceForge.net. http://sourceforge.net. Acesso em: 31/07/2008, 2008. 25

[Sta83] Richard Stallman. About the GNU Project - Initial Announcement. http://www.gnu.org/gnu/initial-announcement.html. Acesso em: 12/08/2011, 1983. 5

[Sta97] Richard Stallman. Linux and the GNU Project. http://www.gnu.org/gnu/linux-and-gnu.html. Acesso em: 02/02/2009, 1997. 93

[Sta02] Richard Stallman. Linux, GNU and Freedom. http://www.gnu.org/philosophy/linux-gnu-freedom.html. Acesso em: 10/02/2009, 2002. 93

[Sta04] Richard Stallman. Free but Shackled - The Java Trap. http://www.gnu.org/philosophy/java-trap.html. Acesso em: 12/05/2011, 2004. 97

[Sun06] Sun Microsystems. Free and Open Source Licensing, Dezembro 2006. 45

[Sup05] Supreme Court of the United States. Metro-Goldwyn-Mayer Studios, Inc. v. Grokster. http://www.eff.org/IP/P2P/MGM_v_Grokster/04-480.pdf. Acesso em: 02/05/2010, 2005. 62

[Swe] David Sweet. The KDE/Qt License History. http://tinf2.vub.ac.be/~dvermeir/manuals/KDE20Development-html/ch19lev1sec4.html. Acesso em: 23/02/2011. 92

[Tau04] Cezar Taurion. *Software Livre: Potencialidades e Modelos de Negócio*. Brasport, Rio de Janeiro, 2004. 5

[The10] The Linux Foundation. Self-assessment checklist, Nov 2010. 85

[Tor91] Linus Torvalds. Notes for linux release 0.01. http://www.kernel.org/pub/linux/kernel/Historic/old-versions/RELNOTES-0.01. Acesso em: 18/04/2011, 1991. 93

[Tor02] Linus Torvalds. *Just for Fun: The Story of an Accidental Revolutionary*. Collins Business, 2002.

[Tor06] Linus Torvalds. Re: GPL V3 and Linux - Dead Copyright Holders (in: linux-kernel list). http://lkml.indiana.edu/hypermail/linux/kernel/0601.3/0559.html. Acesso em: 20/05/2011, 2006. 94

[Tur04] David Turner. The LGPL and Java. http://www.gnu.org/licenses/lgpl-java.html. Acesso em: 16/02/2011, 2004. 72

[Uni94] United States Court of Appeals. Apple Computer, Inc. v. Microsoft Corporation. http://bulk.resource.org/courts.gov/c/F3/35/35.F3d.1435.93-16883.93-16869.93-16867.html. Acesso em: 01/02/2010, 1994. 20

[Uni96] United States Court of Appeals. Lotus Dev. Corp. v. Borland Int'l, Inc. http://bulk.resource.org/courts.gov/c/F3/49/49.F3d.807.93-2214.html. Acesso em: 01/02/2010, 1996. 20

[Uni01] United States Court of Appeals. A&M Records, Inc. v. Napster, Inc. http://bulk.resource.org/courts.gov/c/F3/239/239.F3d.1004.00-16403.00-16401.html. Acesso em: 01/02/2010, 2001. 62

[U.S76] U.S. Copyright Office. Copyright Law of the United States of America. http://www.copyright.gov/title17/92chap1.html. Acesso em: 06/01/2011, 1976. 68

[Was11] Anthony Wasserman. How the internet transformed the software industry. *Journal of Internet Services and Applications*, 2:11–22, 2011. 10.1007/s13174-011-0019-x. 5

[Web04] Steven Weber. *The Success of Open Source*. Harvard University Press, Cambridge, 2004. 12, 13

[Wil09] Andrew Wilson. License Discuss Mailing List: GPL with the Classpath exception - clarification needed. http://www.crynwr.com/cgi-bin/ezmlm-cgi?3:mss:16204:200903:iecmnocjngappddkfkcd. Acesso em: 17/02/2011, 2009. 73