DANIEL NIEBLE DE FREITAS

METODOLOGIA DE DESENVOLVIMENTO DE SOFTWARE LIVRE COM ARQUITETURAS ORIENTADAS A SERVIÇOS: UM ESTUDO DE CASO EM UM AMBIENTE DE TRADUÇÃO AUTOMÁTICA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientadora: Prof. Dra. Laura Sánchez García

CURITIBA

2009

DANIEL NIEBLE DE FREITAS

METODOLOGIA DE DESENVOLVIMENTO DE SOFTWARE LIVRE COM ARQUITETURAS ORIENTADAS A SERVIÇOS: UM ESTUDO DE CASO EM UM AMBIENTE DE TRADUÇÃO AUTOMÁTICA

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientadora: Prof. Dra. Laura Sánchez García

CURITIBA

2009

Agradecimentos

Dentre todas as pessoas que participaram direta ou indiretamente desta conquista, cito um agradecimento especial

Primeiramente aos meus pais (Plínio e Marcia) que me deram toda base educacional que me permitiu ter uma boa formação, ingressar em uma universidade federal e a partir daí ter todas as condições de realizar um curso de mestrado. Em especial a minha mãe que presenciou toda a jornada desde minha graduação até a defesa da banca para o mestrado por quase sete anos e chorou comigo depois da conquista.

À minha noiva (Thays), companheira desde o término da graduação e também se fez presente sempre que foi preciso um incentivo, um carinho e um colo ou apenas compreensão pelo estresse ou cansaço que um mestrado ocasiona.

À meus amigos e colegas que foram a maior base de incentivo para seguir para a área acadêmica e da realização de um mestrado. Em especial meu grande amigo Juliano Picussa o qual devo grande parte dessa vitória por toda a ajuda e ensinamentos, o tenho como amigo e mentor, e nessa jornada não foi diferente... obrigado amigo.

Aos professores da graduação e mestrado do departamento de informática da UFPR, os quais me deram toda base de conhecimento na área de computação, em especial minha orientadora Laura que avaliza essa conquista.

Por fim, o maior de todos os agradecimentos, a Deus, pois a vida só tem sentido por Ele, nada conseguiria se Ele não permitisse e de nada valeria se sem Ele eu vivesse.

Sumário

Lista de Figuras

Glossário de Siglas

SOA	*Service Oriented Architecture*
CVS	*Concurrent Version System*
SQA	*Software Quality Assurance*
XML	*Extensible Markup Language*
SOAP	*Simple Object Access Protocol*
W3C	*World Wide Web Consortium*
WSDL	*Service Web Definition Language*
GPL	*GNU Public License*

Resumo

Este trabalho descreve a composição de uma metodologia de desenvolvimento de software livre na hipótese de um sistema de arquitetura orientada a serviços. Com o intuito de atender à necessidade descrita, partiu-se do contexto de uma arquitetura orientada a serviços, que possui características vantajosas para software livre, e foi construída, a partir de soluções parciais selecionadas da literatura pertinente, uma metodologia de desenvolvimento de software livre para hipótese de trabalho. Por último, a metodologia construída foi aplicada num sistema específico de tradução automática, mais especificamente num ambiente de máquinas de tradução, que podem receber suas bases de conhecimento de qualquer outro sistema, alimentando os léxicos e as regras sintáticas e de mapeamentos inter-línguas necessários para realizar a tradução entre pares de línguas quaisquer. O Opentrad Apertium se apresenta como a máquina de tradução automática em constante evolução, com uma interface especializada para usuários de Computação. Essa ferramenta foi estendida com o objetivo de alimentar o estudo de caso do presente trabalho, de forma a tornar possível a importação de bases de conhecimento originadas de outras ferramentas, por meio de serviços Web.

Abstract

This paper describes the composition of a methodology for developing free software in the event of a system for the services oriented architecture. In order to meet the need described, it was a context of a services oriented architecture, which has advantageous features for free software, and was constructed from partial solutions of selected relevant literature, a methodology of development of free software to a working hypothesis. Finally, the methodology was applied in a system built that consisted of a specific system of machine translation, especially in an environment of machine translation, which can receive the base of knowledge of any other system, feeding the lexical and syntactic rules and inter-language mappings necessary to perform the translation between any pair of languages. The Opentrad Apertium is presented as a machine translation in constant evolution, with a specialized interface for users of computing. This tool has been extended with the aim of feeding the case study of this work, to make possible the importation of bases of knowledge originating from other tools through Web services

Capítulo 1

Introdução

Neste primeiro capítulo será apresentada uma descrição do contexto em que se insere a dissertação, contemplando o problema a ser resolvido, os principais objetivos traçados e a justificativa do estudo. Ao fim do capítulo são descritos a organização e o conteúdo dos capítulos do restante to texto.

1.1. Problema

Apesar de cada vez mais o Inglês ser de fato a língua universal na comunicação entre as pessoas, o respeito à língua original, parte importante da cultura dos povos, faz com que haja uma necessidade cada vez maior de se permitir a comunicação entre as diversas línguas existentes no mundo [1].

As Máquinas de Tradução surgiram como resposta à necessidade de tradução entre diferentes línguas do mundo para proporcionar mecanismos de insumo às ferramentas de tradução automática. Dentre algumas máquinas de tradução existentes no ambiente Web desenvolvidas em software livre, destaca-se o OpenTrad Apertium, um projeto já maduro que se encontra em constante evolução. Ele é uma máquina de tradução automática que consiste em traduzir um texto de uma língua (língua fonte) para outra língua (língua alvo), automaticamente e de maneira a gerar resultados inteligíveis e que preservem as principais características das línguas, como estilo, coesão e significado [2].

Um dos principais obstáculos para a popularização do projeto Apertium é que sua interface usuário para a especificação e a manutenção do conhecimento sobre os pares de línguas e sobre as regras de tradução ainda focam usuários especialistas na área de Computação. Esse obstáculo dificulta a evolução do conhecimento sobre pares de línguas, na extensão das bases, assim como a criação de bases de conhecimento para novos pares de línguas ainda não tratadas pela ferramenta, afetando em ultima analise, a própria robustez do sistema. A arquitetura do sistema Apertium não sendo orientada a serviços, como será descrito aqui nessa dissertação, apresenta algumas oportunidades ainda não trabalhadas com relação ao seu uso e adaptação.

No desenvolvimento continuado que caracteriza ferramentas como esta, no contexto de software livre, uma metodologia deve ser utilizada. Não existe um padrão para o processo de desenvolvimento de software livre, nem uma metodologia bem estabelecida e aceita nesses ambientes ou na Engenharia de Software. Nesta situação, torna-se relevante levantar metodologias disponíveis e propor uma que se adeque às principais características do problema de hipótese.

1.2. Objetivos

1.2.1. Objetivo Geral

O objetivo da presente dissertação consiste na proposição de uma metodologia de desenvolvimento para ambientes em software livre com arquitetura orientada a serviços.

1.2.2. Objetivos Específicos

A consecução do objetivo geral determina a necessidade de atendimento aos seguintes objetivos específicos:

- Analisar diferentes processos e métodos de desenvolvimento de software livre conhecidos e utilizados na comunidade para diferentes casos, identificando suas principais características, artefatos, casos indicados, vantagens e desvantagens;
- A partir do levantamento e das conclusões obtidas dessa análise, aproximando as características identificadas com o estudo de caso de uma máquina de tradução e com a hipótese de arquitetura orientada a serviços, compor uma metodologia de desenvolvimento de software livre, a ser utilizada no desenvolvimento da solução para o caso de estudo;
- Utilizar a metodologia proposta para desenvolver um software que seja uma extensão do software Apertium, para que cada módulo previsto nessa ferramenta possa ser passível de integração e reuso em qualquer outro aplicativo, utilizando serviços Web, além de criar um serviço para a importação de bases de conhecimento entre pares de línguas, parte de interesse e uso de outro trabalho de pesquisa desenvolvido em paralelo.

1.3. Justificativa

Nos últimos anos, um número significativo de publicações na área de Engenharia de Software, mais especificamente sobre software livre, tem buscado descrever o processo de desenvolvimento de forma bem detalhada. No entanto, grande parte destes trabalhos não provê uma visão completa do cenário de projeto de software livre, concentrando-se ou em projetos proprietários ou em aspectos particulares do processo de software.

A crescente adesão à filosofia de software livre, juntamente com o aumento e a melhoria de ambientes de construção colaborativa de conhecimento via Web, exigem uma qualidade cada vez maior, determinando a necessidade de adoção de processos de desenvolvimento adequados. Portanto, a composição de uma metodologia específica para o desenvolvimento de software livre se justifica como área de interesse da engenharia de software.

Por outro lado, o interesse de utilização de software livre no contexto da Lingüística Computacional permite que comunidades de lingüistas e de desenvolvedores possam cooperar entre si, dados seus respectivos legados e conhecimentos específicos, com o intuito de que os projetos produzam aplicações nesse contexto cada vez melhores e mais eficazes.

O estudo de caso aqui descrito consiste em estabelecer abertura para o surgimento de novas frentes de tradução entre variados pares de línguas, desenvolvidas em software livre, em particular daquela cujo ambiente de interface-usuário se constitui no objeto de estudo de outro trabalho em desenvolvimento. Para isso, a ferramenta escolhida para ser estudada e estendida, o Apertium, permite que a tradução seja feita em etapas independentes e que se comunicam por interfaces bem definidas, que podem ser substituídas por outras que obedeçam a essas mesmas interfaces, sem alterar o funcionamento da tradução. Para atender a esse funcionamento, a arquitetura orientada a serviços se faz necessária pelas suas características de reuso e coesão. Essa adaptação da arquitetura do sistema para o paradigma orientado a serviços faz com que o sistema se torne acessível de maneira modularizada através da Web, alem de permitir facilmente a portabilidade com outras plataformas pelas características intrínsecas dessa arquitetura.

Neste contexto, essa dissertação compõe uma metodologia de desenvolvimento para software livre que contempla aspectos intrínsecos da arquitetura orientada a serviços, e aplica ao estudo de caso da extensão da ferramenta Apertium, como forma de ilustrar o seu em um caso concreto, contribuindo, adicionalmente, com o ambiente colaborativo de tradução automática.

1.4. Estrutura da Dissertação

O restante da presente dissertação está dividido da seguinte forma.

No capítulo 2 é apresentado um breve histórico das ferramentas de tradução, contendo sua evolução e principais características que levam a entender de modo geral o contexto e alguns aspectos da solução proposta no presente trabalho. O capítulo 3 consiste numa resenha literária sobre trabalhos relacionados, que utilizam metodologias de desenvolvimento de projetos em software livre. O capítulo 4 descreve em linhas gerais a arquitetura Orientada a Serviços, hipótese do problema. No capítulo 5 são descritos a metodologia de desenvolvimento composta neste trabalho e seu uso aplicado ao estudo de caso mencionado. O capitulo 6 relata as conclusões sobre o trabalho e apresenta propostas de pesquisas futuras.

Capítulo 2

Tradução Automática

A Tradução Automática é um ramo da Lingüística Computacional. Iniciada na década de 1940 teria sido a primeira aplicação não numérica da Computação e a partir da década de 1960, vem se desenvolvendo de forma significativa tanto em número de publicações de pesquisas quanto no mercado de software [6].

O objetivo inicial desta disciplina foi simular a tradução feita pelo homem, por meio dos Sistemas de Tradução Auxiliada por Humanos (*Human-Aided Machine Translation*) e de Tradução Humana Auxiliada por Máquinas (*Machine-Aided Human Translation*). Esses projetos de tradução não sofreram muita evolução, não tendo atingido resultados significativos, nem sido utilizados na prática, assim sendo abandonados [3].

A tradução automática foi vista como algo impossível, no Relatório ALPAC pela Academia de Ciências Americana em 1966. Neste documento, havia explicações e conclusões negativas com relação às chances de sucesso da tradução automática [7].

Mesmo nos sistemas completamente automatizados, era necessário fazer algumas adaptações e simplificações, por exemplo, ao considerar um subconjunto menor do domínio completo das línguas por meio da exclusão de algumas construções sintáticas ou de alguns tipos de sentença, determinando, assim, subconjuntos próprios passíveis de tradução.

Atualmente há quatro abordagens principais para a tradução automática: 1) uma que usa exclusivamente o conhecimento lingüístico, baseado em dicionários e gramáticas (*Language-Based Machine Translation*); 2) uma baseada em conhecimentos gerais de Lingüística, como dicionários, gramáticas, enciclopédias e bases de conhecimento (*Knowledge-Based Machine Translation*); 3) outra baseada em probabilidades *(Statistics-Based Machine Translation)* e 4) uma última baseada em exemplos *(Example-Based Machine Translation).* As duas primeiras abordagens são mais ligadas a estruturas lingüísticas associadas ao processo de tradução dos pares de línguas considerados, e geralmente usam regras para explicitar o conhecimento lingüístico desse par de línguas. As duas últimas utilizam modelos matemáticos de Probabilidade e Estatística [3].

Até hoje as máquinas de tradução mais utilizadas são proprietárias, não livres, tanto do ponto de vista de sistema como das suas base de conhecimento, como exemplo os tradutores *Trados Workbench* (http://www.trados.com/), *IBM TranslationManager* (http://www-4.ibm.com/software/ad/translat/), *Déjavu* (http://www.atril.com) e a usada pela Xerox *Systran* (http://www.systransoft.com), criado inicialmente para traduzir seus manuais técnicos em várias línguas. Estes pacotes de software são distribuídos comercialmente ou acessíveis pela Internet seguindo o mercado virtual [37].

Mesmo usuários mais experientes podem apenas utilizar os produtos, pois normalmente não são passíveis de adaptação a necessidades particulares, de integração a projetos de pesquisas, de aplicação ou mesmo de uso como contribuições para o desenvolvimento de tecnologias já existentes, pois o código não é acessível ou a a licença não o permite.

A maneira como estes sistemas comerciais são distribuídos tem um impacto negativo para o desenvolvimento de novas técnicas, para a ampliação do conhecimento sobre tradução de línguas e para a criação de novos pares de tradução menos atrativos comercialmente.

Impulsionada pela Internet na ultima década, as comunidades de software livre aprimoraram estratégias para o desenvolvimento de software com reutilização de código e conhecimento guardados em repositórios unificados e compartilhados. Com base nesses princípios, os pesquisadores puderam focar seus esforços em suas próprias áreas visando o contínuo aprimoramento do software já desenvolvido. Neste cenário começaram a nascer e coexistir máquinas de tradução entre línguas alternativas àquelas normalmente consideradas pelo software proprietário.

Uma máquina de tradução em software livre pode ser usada, copiada, estudada, modificada e distribuída, podendo não haver nenhuma cobrança ou custo para sua aquisição. As vertentes do software livre podem ser aplicadas a elementos distintos do software sempre que o código fonte estiver disponível no produto final. Para que um sistema de tradução automática se considere um software livre, é necessário que o código fonte do motor de tradução e dos dados lingüísticos (bases de conhecimento) sejam distribuídos e abertos. Deve ser considerado que é mais provável que os usuários que queiram alterar os códigos fonte desse sistema o façam mais nos dados lingüísticos do que no motor de tradução em si [4].

2.1. Apertium

O OpenTrad Apertium, mencionado na seção 1.1., é um sistema de tradução automática completo e em uso. O projeto de implementação desse sistema é mantido no repositório de software livre, SourceForge, publicado sob os termos da licença GNU GPL, a

ser explicado detalhadamente ao longo dessa dissertação, e coordenado principalmente pelo grupo Traducens da Universidade d`Alacant, Espanha, que também o idealizou. Esse projeto fora inicialmente desenvolvido para tradução entre pares de línguas de origem comum, e se encontra atualmente em sua versão 3.0, permitindo o desenvolvimento de pares de tradução entre quaisquer pares de linguagem que possam ser formalizadas em forma de regras sintáticas. A ferramenta Apertium foi desenvolvida majoritariamente na linguagem C++ e pode ser compilada e executada sobre o sistema operacional Linux. Apesar de não ter muita portabilidade ainda para outros sistemas operacionais, suas funcionalidades podem ficar acessíveis instalando a ferramenta em um servidor de aplicação na Internet e acessando-o de forma remota [4].

Para produzir rapidamente, de forma inteligível e de fácil correção às traduções entre línguas relacionadas, o Apertium utiliza um clássico sistema de tradução indireta que consiste na transferência parcial sintática palavra a palavra, uma estratégia semelhante à utilizada em máquinas de tradução comerciais conhecidas [5].

Os arquivos utilizados pelo Apertium utilizam a o formato XML(*eXtensible Markup Language*), devido à sua interoperabilidade e à sua independência em relação ao conjunto de caracteres a ser considerado, assim como à disponibilidade de ferramentas e bibliotecas já existentes, o que torna mais fácil a análise dos dados nesse formato. O formato XML é um padrão de representação de dados na Web, contendo diversas tecnologias que o utiliza e possuindo avançados mecanismos de acesso e edição de códigos nesse formato [5].

A arquitetura do Apertium é dividida em 8 módulos que funcionam de forma articulada, associando o conceito de serviços disponíveis e que possuem pouca dependência entre si. Essa divisão modular da arquitetura é apresentada na figura 1 e descrita na seqüência.

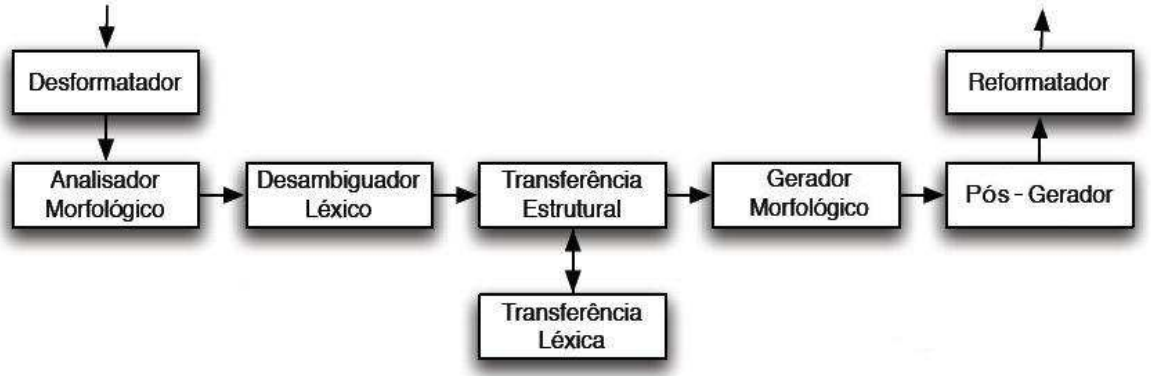

Figura 1: Módulos da Arquitetura do Apertium [17]

- **Desformatador**: separa o texto a ser traduzido do formato original (RTF, HTML, etc.) em que o documento se encontra, as informações da formatação são encapsuladas, para que os próximos módulos as tratem como espaços em branco entre as palavras, um exemplo seriam as *tags* da linguagem HTML;
- **Analisador Morfológico**: segmenta o texto morfologicamente e retorna uma ou mais formas léxicas. Cada parte separada é composta por um lema (forma base da palavra comumente utilizada nos dicionários clássicos), uma categoria morfológica (substantivo, verbo, preposição, entre outros) e informações de inflexão morfológica (número, gênero, pessoa, etc.);
- **Desambiguador Léxico**: retira as ambigüidades encontradas, elegendo (estatisticamente) uma das formas léxicas de acordo com o seu contexto, quando o Analisador Morfológico encontrar mais de uma.
- **Transferência Estrutural**: detecta padrões de palavras que precisem de um tratamento especial em virtude de divergências estruturais entre o par de línguas envolvido (trocas de gênero, trocas de número, entre outros).

- **Transferência Léxica**: funciona com base num dicionário bilíngüe e é invocado pelo módulo de Transferência Estrutural, conforme apresentado na figura 1. Este módulo recebe uma forma léxica na língua original e retorna uma forma léxica correspondente na língua de destino. Aqui é feito o processo de tradução propriamente dito, de acordo com as regras de tradução para o par de línguas considerado.
- **Gerador Morfológico**: constrói a forma superficial adequada para cada uma das formas léxicas já na língua destino, usando para isso novamente um Analisador Morfológico.
- **Pós-Gerador**: realiza algumas operações ortográficas simples na língua de destino, como contrações ou colocação de apóstrofos.
- **Reformatador**: gera o texto novamente, agora traduzido na língua destino, em formato equivalente ao formato original.

A plataforma Apertium foi estendida com base na metodologia de desenvolvimento em software livre focada para arquitetura orientada a serviços proposta, a título de estudo de caso de aplicação dessa metodologia. O estudo de caso consiste em permitir o uso de módulos de seu funcionamento por outras ferramentas, por meio de integrações por serviços Web, e, em particular, a alteração de suas bases de conhecimento com a importação de uma base adquirida da execução de outras ferramentas, objeto de uma dissertação complementar. O resultado da aplicação do estudo de caso ainda poderá ser utilizado como teste em trabalhos futuros, fazendo com que o Apertium receba os léxicos e as regras geradas por um ambiente de especificação do conhecimento sobre pares de línguas quaisquer e execute a tradução.

Capítulo 3

Metodologia de Desenvolvimento em Software Livre

Neste capítulo serão apresentados trabalhos disponíveis na literatura sobre metodologias de desenvolvimento de projetos em software livre, juntamente com uma análise de cada uma e a indicação das situações para as quais elas são apropriadas.

O software livre é uma realidade na literatura científica nos diversos ramos de pesquisa em Computação. O objetivo geral da presente dissertação consiste em descrever uma metodologia de desenvolvimento de software que possa para ser usada no processo de desenvolvimento para uma plataforma colaborativa de construção de conhecimento, em particular para máquinas de tradução.

3.1. Software Livre

Antes de apresentar e analisar os processos de desenvolvimento de software livre serão descritas abaixo as terminologias relacionadas a software livre, mais especificamente a sua definição relacionada à propriedade intelectual e licenças.

O conceito de licença de software determina a forma como um software pode ser usado pela comunidade. Essa licença se constitui integralmente pelos regimentos das leis de copyright (conjunto de leis que se referem a propriedade intelectual) presente em um documento junto ao software, apresentando explicitamente em quais condições pode ser

utilizado [38]. A lista de categorias de software descrita em *Categories of Free and Non-Free Software* do *Free Software Foundation* inclui um grande número de tipos diferentes, tais como [20]:

- **Software Proprietário**: software que proíbe a redistribuição e alteração pelo usuário, apenas o uso através da aquisição do mesmo.
- ***Freeware:*** software que permite redistribuição, mas não qualquer modificação nos códigos-fonte e geralmente nem há acesso a esses fontes.
- ***Shareware*****:** software que permite redistribuição, mas que restringe o uso de acordo com uma condição específica, normalmente associada à um tempo limite de uso.
- **Software Livre:** software que oferece ao usuário o direito de usar, estudar, modificar e redistribuir os códigos-fonte de qualquer maneira e em qualquer tempo.

Dentro dos softwares considerados livres pela definição acima existem diversas licenças, cabe citar a mais importante delas, responsável pelo licenciamento de 70 % dos softwares livres, a GNU GPL**,** que é não-permissiva, ou seja, permite redistribuição apenas se for mantida a garantia de liberdade aos receptores da cópia redistribuída, também obrigando versões modificadas a serem também livres, acompanhadas de seus fontes.

3.2. Processos de Desenvolvimento em Software Livre

É notória a diferença entre trabalhar com equipes próximas, bem supervisionadas, com regras e procedimentos estabelecidos por organizações, quando no desenvolvimento de software comercial, e realizar o desenvolvimento de software livre, em que normas e regras são bem flexíveis e podem ser adequadas quando uma particularidade mostra-se interessante e necessária.

A metodologia de desenvolvimento de software livre pode ser entendida como um processo de trabalho virtual, por não possuir estatutos ou regras pré-estabelecidas, diferentemente das metodologias adotadas em empresas e instituições que desenvolvem software. Também não há, em software livre, compromisso algum de confidencialidade. “Um Projeto de Software Livre é uma organização composta por um conjunto de pessoas que usa e desenvolve um único software livre, contribuindo para uma base comum de código-fonte e conhecimento. Este grupo terá à sua disposição ferramentas de comunicação e trabalho colaborativo, e um conjunto de experiências e opiniões técnicas que usam para discutir o andamento do projeto.” [29].

Os projetos de desenvolvimento de sistemas em software livre possuem algumas características particulares que os diferenciam de projetos convencionais, que são:

- **Trabalho Voluntário**: a motivação financeira normalmente não é o foco principal para os desenvolvedores, o que representa uma grande diferença com relação ao desenvolvimento do software comercial. As pessoas envolvidas têm interesse na estrutura interna e na forma em que o software é construído, assim como contribuem sempre de acordo com sua disponibilidade, o que é efetivado, por exemplo, por meio de discussões por correio eletrônico sobre os assuntos relacionados ao software em questão. Não é possível estabelecer sempre o número exato de pessoas que fazem parte de um projeto em Software Livre, pois colaboradores podem perder interesse ou não dispor de tempo para promover seu desenvolvimento, assim como novos colaboradores podem surgir a qualquer momento. O gerenciamento de recursos para o projeto desse software deve considerar esses aspectos, pois não existem horários ou listas de entrega. Como o trabalho é voluntário, as pessoas fazem seus horários e contribuem quando e como

podem, visto que normalmente não possuem um compromisso formal ou mesmo financeiro com o desenvolvimento do sistema;

- **Trabalho não é designado**: de modo geral, as tarefas não são impostas a ninguém, as pessoas trabalham em partes do projeto nas quais têm mais aptidão, interesse. Em alguns grupos de desenvolvimento em software livre, as tarefas podem ser designadas e cobradas de acordo com um planejamento e experiência de cada um;
- **Versões**: como o trabalho é voluntário e não há compromissos com prazos, o lançamento do produto ocorre quando a equipe de coordenação do projeto acha que está pronto para ser usado e exposto a críticas;
- **Testes**: uma vez que muitas pessoas podem ter desenvolvido partes de um módulo do sistema, ou mesmo podem ocorrer muitas evoluções do mesmo software, a fase de testes é essencial para a garantia do funcionamento correto do código que foi implementado ou alterado, antes de liberar uma versão para a comunidade e validação dos usuários finais (homologação). A fase de testes contempla normalmente dois tipos: testes unitários e testes de sistema. Testes unitários validam partes individuais do sistema (funções, módulos, telas entre outros) e testes de integração verificam o funcionamento destas partes combinadas, incluindo outros aspectos como desempenho, adequação e consistência da funcionalidade com a especificação.
- **Planejamento**: esta atividade em software livre é a que mais se diferencia do processo de desenvolvimento de software comercial, devido à falta de compromisso e de certeza da colaboração dos desenvolvedores, dos testadores e dos usuários.

A participação dos colaboradores nesses projetos ocorre em diferentes níveis de participação e envolvimento: [20]

- **Colaboradores não Participantes**: usuários finais apenas utilizam o software, mas não contribuem para sua melhoria;
- **Colaboradores Participantes**: contribuem ativamente para o projeto, informando erros e discutindo possíveis melhorias nas funcionalidades e estruturação e arquitetura da solução;
- **Colaboradores Esporádicos**: contribuem de forma inconstante na evolução do projeto, às vezes apenas respondendo a questões de listas de discussão ou dando pequenas sugestões no andamento do projeto;
- **Equipe de Coordenação**: esta equipe é responsável pelo gerenciamento do projeto e pela tomada de decisões de cronograma e principalmente pelo lançamento de novas versões, além de organizar os colaboradores nas tarefas que vão executar no planejamento do projeto.

Não existe uma metodologia padrão de desenvolvimento de software livre. Portanto é preciso investigar algumas práticas de trabalho de diferentes comunidades envolvidas com esse tipo de software, para ser capaz de identificar as melhores características de cada uma e compor uma sugestão de metodologia de trabalho própria.

3.3. A Catedral e o Bazar

Eric Raymond expressa no artigo *The Cathedral and The Bazaar* [31] a existência de uma divisão clara entre duas formas distintas de desenvolvimento de software. A catedral, que descreve projetos de software desenvolvidos por grupos fechados com fins mais comerciais, com pequena ou nenhuma abertura para participação externa; e o bazar, que descreve projetos desenvolvidos de forma mais visível, disponíveis à colaboração de qualquer pessoa que tenha interesse em dar a sua contribuição.

A catedral, para Raymond, é uma forma conservadora e tradicionalista de desenvolvimento, pois ilustra a criação de uma grande obra de engenharia, onde um grupo pequeno de sábios projeta e refina uma construção sobre a qual eles têm conhecimento. Ele destaca nessa forma longos ciclos de desenvolvimento e grande demora entre lançamentos de versões. O bazar, por sua vez, é descrito como um projeto colaborativo e aberto ao ponto da "promiscuidade", pois qualquer usuário ou colaborador é livre para participar e opinar sobre o desenvolvimento e sobre o produto. Nesse modelo, os lançamentos ocorrem freqüentemente e muita gente pode ter acesso ao código-fonte, permitindo que erros ou melhorias sejam rapidamente avaliados e informados.

O artigo coloca alguns requisitos que devem ser atendidos em qualquer metodologia que vise um projeto de software na metáfora do bazar. Raymond defende a importância de ter uma grande base de usuários com acesso ao código-fonte do ponto de vista da garantia da qualidade. O autor também defende a prática de ter sempre disponível, para os usuários e outros desenvolvedores, a oportunidade de testar freqüentemente a base de código mais recente.

3.4. Modelo de Desenvolvimento do S.O. Linux

A figura 2 apresenta o processo de desenvolvimento do sistema operacional Linux, que é um software livre de grande impacto, com milhares de desenvolvedores espalhados pelo mundo todo, conseguindo manter sua estabilidade. Suas atividades são focadas nas etapas de codificação e testes, que têm bastantes interações entre si e que juntas evoluem para o surgimento de novas versões. [18]

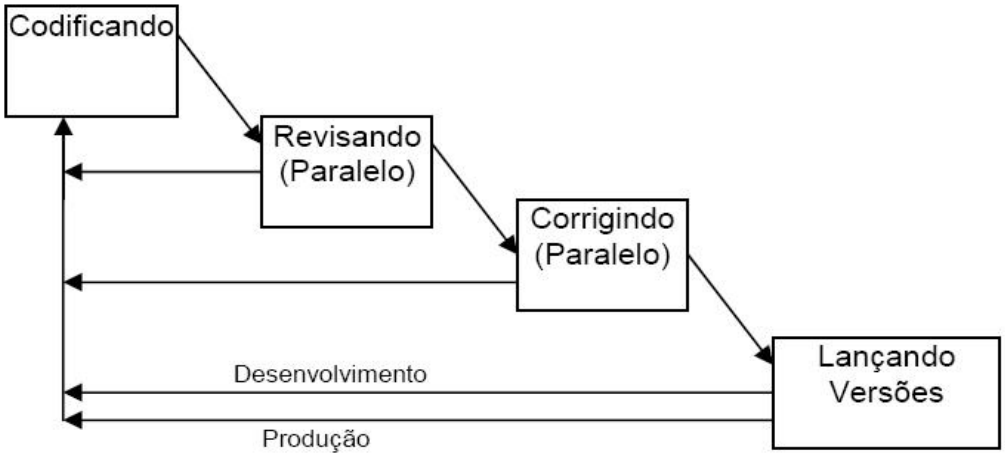

Figura 2: Modelo de Desenvolvimento do Sistema Operacional Linux [18]

As atividades consistem em pequenas correções ou melhorias para o software disponível na comunidade e, assim, a quantidade de código modificada em cada etapa é menor, facilitando o retorno tanto de usuários quanto de outros desenvolvedores que colaboram na sua evolução. Essas atividades seguem um modelo em que cada desenvolvedor age baseado em seus conhecimentos e experiência e na sua própria intuição, não recebendo ordens ou instruções de outras pessoas.

Existe ainda um processo de validação final ou homologação, que quando são de alterações mais críticas ou que envolvam o Kernel (núcleo), é realizado exclusivamente pelo próprio Linus Torvalds, idealizador e desenvolvedor principal do Linux.

O Kernel do Linux consiste em uma série de arquivos escritos tanto em linguagem C e quanto em linguagem Assembly que constitui o núcleo do sistema operacional. Ele pode ser visto como uma interface entre os programas e todo o hardware, alem de permitir que todos os processos sejam executados pela CPU e também que estes consigam compartilhar a memória do computador, pois é ele que controla todo o hardware do computador [32].

3.5. Modelo de Desenvolvimento do Apache

O projeto Apache foi um dos pioneiros na definição de processo de desenvolvimento do software livre, pois existiu antes mesmo de gerar qualquer linha de código. Como o time de desenvolvedores estava distribuído ao redor do mundo e a grande maioria dos desenvolvedores possuía um outro emprego e se dedicava ao projeto em diferentes horários do dia, um processo único de desenvolvimento era necessário para permitir a tomada de decisões durante o projeto. Assim, surgiram como procedimentos típicos o uso de ambientes e meios de comunicação, tais como os grupos e as listas de discussão por correio eletrônico. Os primeiros desenvolvedores criaram o AG (*Apache Group* - Grupo Apache). Existiam alguns desenvolvedores principais do projeto que tinham mais responsabilidades e, assim, tinham permissão para acrescentar código definitivo ao projeto. Estes também aceitavam ou reconheciam novos membros e votavam em mudanças no código principal que pudessem influenciar o trabalho de outros desenvolvedores [21].

O processo de evolução do software consistia no seguimento por cada desenvolvedor, de um conjunto de ações padronizadas e documentadas para o projeto. Essas ações eram: 1) descobrir um problema ou selecionar um já conhecido; 2) determinar um voluntário para trabalhar nele ou voluntariar-se; 3) identificar uma solução para o problema; 4)desenvolver o código e testar em uma cópia ou base local do sistema; 5) apresentar as modificações efetuadas para o AG revisar e enviar o novo código alterado, devidamente documentado ao repositório principal. Nesta fase de evolução, várias solicitações eram feitas e enviadas por meio das listas de discussão ou de uma ferramenta de controle de erros e alterações, como o Bugzilla.

A comunidade Apache utilizou o Bugzilla que é uma ferramenta de controle de alterações implementada em uma aplicação Web que corresponde a uma importante parte do conjunto de ferramentas de desenvolvimento construído para apoiar o desenvolvimento do projeto Mozilla, que é um outro importante projeto em software livre para desenvolvimento de navegadores para uso da Internet. [28]

Uma alteração de código, uso, documentação ou licença tem início no momento em que é identificada e solicitada por um usuário do sistema. O Bugzilla utiliza a solicitação da alteração como elemento fundamental e associa a ela um número seqüencial. Cada um desses registros no controlador de alterações possui os seguintes itens de informação [28]:

- Descrição do problema ou necessidade identificada;
- Nome e endereço eletrônico de uma ou mais pessoas participantes da alteração, incluindo o solicitante, o responsável pela garantia de qualidade e o desenvolvedor;
- Comentários sobre a procedência e a evolução da solicitação de alteração;

- Grau de prioridade;
- Anexos associados à alteração, como casos de teste e o próprio código-fonte;
- Estado da solicitação, que podem assumir os valores: “Informado”, “Confirmado”, “Em Implementação”, “Finalizado”.

A descoberta de problemas era realizada por meio de envio de mensagens nas listas de correio eletrônico associadas ao desenvolvimento do projeto, usando um sistema de controle de problemas ou por meio de postagem nos grupos de discussão. Listas contendo o estado do projeto, os problemas prioritários, as questões pendentes ou iniciadas, o planejamento de liberação de versões, eram mantidas junto ao repositório do próprio código, a fim de permitir o acompanhamento por todos os interessados e garantir a consistência e a segurança dessa informação, vitais para o processo definido.

No processo descrito para o Apache, uma vez que muitas pessoas podiam estar trabalhando em paralelo desenvolvendo partes do sistema isoladamente, uma preocupação constante no gerenciamento consistia no controle de versões. Com este fim foi utilizada uma ferramenta específica de controle de versões de documentos, o CVS (Concurrent Version System). [19]

O modelo básico de funcionamento do CVS permite trabalho *offline*, tornando possível que o desenvolvedor faça alterações em uma cópia local do código-fonte, sem necessitar estar conectado ao servidor que o hospeda. Além disso, múltiplos desenvolvedores podem editar cópias do mesmo projeto em sua base local de maneira concorrente, e que não atrapalhe ou dependa de outros desenvolvedores.

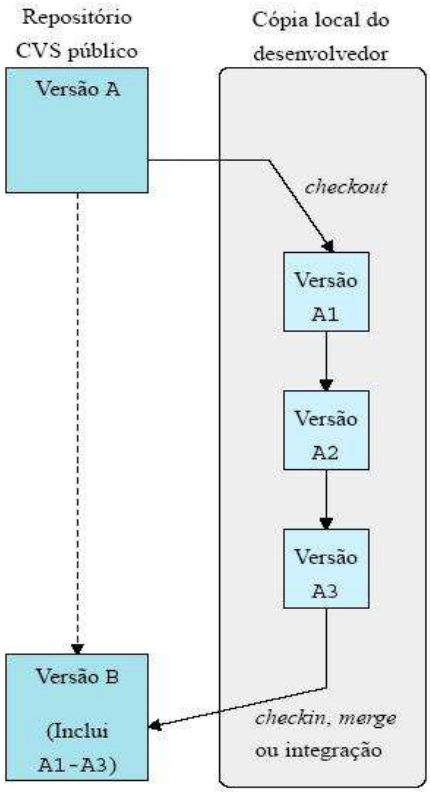

Figura 3: Modelo de Funcionamento do Controlador de Versão CVS [20]

A figura 3 representa de maneira geral como funciona um repositório de um software controlador de versões, no caso o CVS. Uma versão A estável e disponível no repositório é reservada para uso e alteração por um desenvolvedor. Este passa a alterar seu conteúdo gerando outras versões locais (como representado pelas versões A1, A2 e A3), que ficam disponíveis apenas no espaço local da máquina do desenvolvedor e nenhum outro tem acesso a elas. Apenas quando o desenvolvedor libera a nova versão para o repositório, gerando a versão B, esta passa a estar disponível novamente para qualquer outro desenvolvedor utilizar e/ou alterar os arquivos da versão. Durante a etapa em que uma versão está reservada para certo desenvolvedor, os outros podem apenas obter a versão, mas não alterá-la. Assim, o desenvolvedor com a versão reservada sempre estará fazendo uso da última versão disponível no repositório, na figura a versão A.

O projeto Apache possui um sub-projeto chamado de *Apache HTTP Test Project*, responsável pela realização de testes nos módulos do projeto. Todo código enviado para o repositório precisa ser inspecionado pelo processo definido por esse sub-projeto. Essa inspeção é realizada por um grupo especifico, responsável por garantir a qualidade do código que está sendo implementado.

3.6. Outros Trabalhos Relacionados

Os trabalhos mencionados a seguir são referentes a pesquisas relacionadas ao processo de software, considerando o universo do software livre. Assim, para os diversos itens mencionados para estes trabalhos, deve ser considerado o ciclo de vida do software como um todo, e não apenas o software original a partir de sua idealização e primeira liberação para a comunidade.

No artigo de Scacchi [26] foi feita uma comparação entre 4 tipos de softwares livres desenvolvidos para diferentes fins e uma análise dos processos e metodologias utilizados em seus respectivos processos de desenvolvimento. Em contraste com o mundo acadêmico de Engenharia de Software, o desenvolvimento de software livre não adota processos previstos na área de pesquisa em questão. Os estudos revelados no artigo identificaram cinco atividades ou práticas empregadas em todos os processos analisados:

- Análise e especificação de requisitos;
- Coordenação de controle de versões, controle de geração de executáveis, e liberação de versões;
- Manutenções evolutivas, constantes e com respectivas redistribuições;

- Gestão de projetos;
- Uso de software de transferência de tecnologia.

Cada uma das práticas apresentadas difere das tradicionais atividades da Engenharia de Software. Nenhuma delas deve ser entendida como independente ou mais importante do que as outras. A análise feita pelo autor mostra que estas práticas podem ocorrer simultaneamente, completa ou parcialmente, não estritamente seqüencial dentro de um modelo do ciclo de vida tradicional.

O autor conclui que essas práticas do desenvolvimento do software estão dando lugar a uma nova visão de diferentes e complexas formas de construção, implementação e evolução de programas. O desenvolvimento de software livre não aderiu às tradicionais metodologias encontradas no legado do software ou de modelos e normas bem estabelecidas, ou seja, o desenvolvimento de software livre é inerentemente uma complexa teia de processos sócio-técnicos, de dispersas situações e contextos, mas em franco crescimento no mundo todo. Esta posição apresenta um quadro empírico que começa a delinear alguns dos contornos e dinâmicas que caracterizam o modo como os sistemas de software livre e as respectivas comunidades são interligados e alterados em benefício de ambos, sem metodologias convergentes.

O artigo de Reis [20] apresenta uma análise de alguns projetos conhecidos no ambiente se Software Livre do ponto de vista de seus processos de desenvolvimento, avaliando especialmente as atividades realizadas para produzir, gerenciar e garantir a qualidade do software. O conteúdo do trabalho foi resultado da participação ativa do autor no desenvolvimento e na evolução dos próprios projetos, assim como da aplicação de um questionário a um conjunto representativo de líderes e desenvolvedores de cada projeto, que

tinha como objetivo levantar atividades executadas durante todas as etapas do processo de desenvolvimento do software livre. Os projetos de base neste levantamento são projetos representativos como Linux, Mozilla, Perl, MySQL, KDE, diversos projetos GNOME e outros tantos.

A partir da análise do autor dos questionários respondidos, algumas hipóteses referentes a aspectos e a características do desenvolvimento em software livre são provadas, no escopo da realização da referida pesquisa. Essas hipóteses foram elaboradas a partir de um levantamento obtido da literatura existente sobre o tema, e da experiência obtida pelo autor na sua participação nos projetos. A função do questionamento, então, passou a ser confirmar essas hipóteses.

Os resultados desse estudo, em resumo, são:

- **O trabalho é realizado por equipes geograficamente dispersas**: essa hipótese foi confirmada para a maioria dos projetos nos quais, segundo o autor, as pessoas nem chegam a se conhecer pessoalmente. No entanto nos últimos anos, tem havido uma mudança ainda sutil neste perfil. O investimento de empresas e outras organizações convencionais em projetos de software livre e a ocorrência de um número maior de eventos internacionais têm contribuído para aproximar os membros das comunidades;
- **O desenvolvedor do software é também seu usuário e contribui efetivamente para determinar grande parte de sua funcionalidade**: essa hipótese foi confirmada, pois grande parte dos projetos é criada por motivos pessoais, e a grande maioria dos projetos tem como usuários principais sua própria equipe de desenvolvimento;

- **Todo projeto utiliza um sistema de liderança particular para coordenar e arbitrar disputas, existindo duas variantes principais: o 'ditador benevolente' de Raymond, e o grupo central (ou *core*) de desenvolvedores**: a pesquisa confirma que, na maior parte dos projetos, um sistema de liderança é utilizado;
- **Há uso amplo de ferramentas para viabilizar a comunicação entre a equipe geograficamente dispersa**: esta hipótese foi confirmada. Verificou-se a preferência por listas de correio eletrônico, utilizadas pela grande maioria dos projetos. Entretanto, uma outra parcela significativa, indicou utilizar ferramentas de acompanhamento de alterações, como o Bugzilla, mencionado na seção 3.3;
- **Há uso amplo de ferramentas de controle de versão, em particular do CVS**: esta hipótese foi comprovada. A grande maioria dos projetos utilizou um sistema de controle de versões;
- **A maior parte dos projetos possui equipe pequena, e a média do número de indivíduos por equipe não passa de 5**: Esta hipótese também foi comprovada. A tendência é de os projetos possuírem entre 2 e 5 desenvolvedores, e uma parcela considerável possui apenas um. No entanto, o grau de complexidade e mesmo do tamanho do sistema a ser desenvolvido, pode levar a criação de equipes maiores;
- **A maior parte das equipes possui membros com mais de 5 anos de experiência em desenvolvimento de software**: essa hipótese não foi comprovada pela pesquisa e análise do autor;
- **O trabalho de engenharia de requisitos é freqüentemente facilitado por levantamentos anteriores feitos por outras equipes, seja por meio de padrões estabelecidos ou documentados, seja por meio de código-fonte herdado de outro projeto**: segundo o autor da pesquisa, os resultados sugeriram fortemente o

uso de conhecimento pré-existente no estabelecimento dos requisitos dos projetos;

- **Existe um compromisso pessoal por parte dos membros da equipe do projeto em garantir a qualidade do software lançado**: essa hipótese é subjetiva, e portanto, não passível de comprovação ou refutação. Nem mesmo a metade dos projetos efetua teste extensivo quando do lançamento de uma versão publicamente, o que sugere um nível variável de compromisso das equipes com a qualidade do software;
- **Existe uma forte política de controle e revisão relacionada à aceitação, à integração e à auditoria de contribuições de código**: os resultados da pesquisa, surpreendentemente, não comprovaram a execução destas atividades entre os projetos;
- **O tamanho da base de código do projeto está diretamente relacionado à dimensão da equipe e ao tempo de existência do projeto**: esta hipótese foi comprovada pelo autor, pois houve indicação de vínculos entre estas grandezas;
- **A equipe do projeto tende a crescer com o tempo**: os resultados relativos a essa hipótese são considerados não conclusivos pelo autor, dado a grande diversidade dos dados obtidos;
- **Pouca atenção é dada à usabilidade do software**: esta hipótese foi comprovada pelo autor por não ter encontrado essa prática indicada pelos resultados.

O autor apresenta também uma síntese de variantes de processos encontrados nos projetos avaliados e conclui que, de uma maneira geral, o desenvolvimento de software livre parece variar substancialmente entre os projetos, o que, segundo ele, restringe a possibilidade de generalização quando se trata destes ambientes.

No artigo de Nunes e Staa [27] é feito um levantamento de algumas abordagens para o desenvolvimento em software livre e, ao final, é descrito um modelo de processo de software para este tipo de contexto. De maneira geral, o artigo descreve como os passos de um ciclo de vida padrão de projetos de desenvolvimento em software livre os seguintes:

- Um idealizador do projeto escreve uma versão inicial e publica, em algum local na Internet (normalmente repositórios conhecidos da comunidade de software livre), a descrição do software juntamente com seu código fonte;
- Quando o software desperta o interesse de algum desenvolvedor ou mesmo de algum usuário, são propostas modificações e melhorias ao autor;
- O autor recebe essas propostas e as analisa, as filtra e modifica o código fonte do software de acordo com o que ele considera relevante;
- Quando o autor ou não concorda ou não pode alterar o software original, esses são discutidos em fórum aberto pela própria comunidade de software livre. As pessoas envolvidas e interessadas podem então fazer as correções e/ou alterações nesse código, gerando uma nova versão adaptada que é publicada com um novo (ou novos) autores;
- Essa versão melhorada atrai novos usuários, desenvolvedores, novas criticas e sugestões;
- Se o projeto não consegue atrair usuários (colaboradores), ou a versão disponibilizada esta atendendo às necessidades e interesses das pessoas sem necessitar de evoluções, o projeto fica congelado até esta situação se modificar.

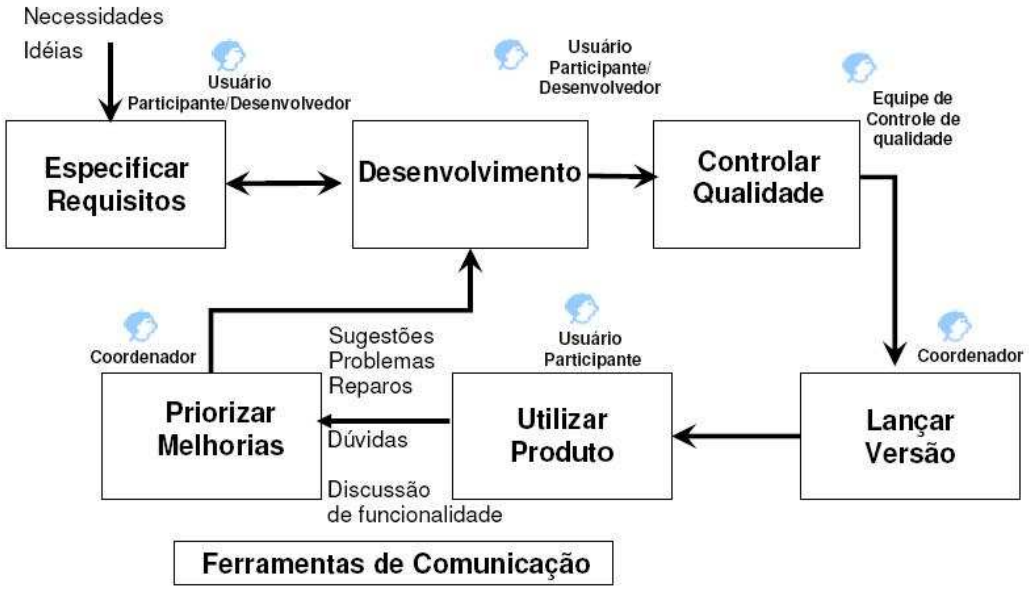

Figura 4: Caracterização do Processo de Software por Nunes e Staa [27]

A figura 4 representa o fluxo de atividades descritos na metodologia de desenvolvimento em software livre proposta no artigo. As fases apresentadas são de:

- **Especificar Requisitos**: como a maioria dos projetos em software livre não possui uma especificação bem delimitada e formal, espera-se que o colaborador não só codifique, como também documente o software seguindo um determinado padrão, incluindo o objetivo da alteração e os requisitos que o levaram a realizar a mesma. Assim busca-se pelo menos ter uma documentação mínima, para, assim, facilitar a evolução do projeto e a entrada de novos colaboradores;
- **Desenvolvimento**: a fase de desenvolvimento ocorre de maneira pouco formal na maioria dos projetos e o tempo de desenvolvimento pode variar muito, de acordo com a disponibilidade e o empenho dos colaboradores, assim como o grau de complexidade e porte dos projetos;

- **Controlar a Qualidade**: nesta fase, uma equipe de controle da qualidade é responsável por avaliar e identificar melhorias na qualidade do software. Em geral, as atividades da equipe se resumiriam na auditoria do código-fonte, uma atividade que algumas metodologias da Engenharia de Software apresenta como *Peer Review*. Nessa revisão de código, três papéis devem interagir: um moderador, os revisores do código e um responsável por documentar seguindo um determinado padrão os resultados da auditoria. Os autores defendem a não participação, nesta fase, dos desenvolvedores do código em avaliação;
- **Lançar Versão**: O processo de lançamento de uma nova versão consiste inicialmente na definição de um responsável, quem terá a função de testar a versão, gerar um rótulo no repositório dos arquivos e ser o mentor da correção dos possíveis erros e das alterações, assim como da especificação de requisitos para a versão. Muitas vezes, existe a necessidade de "empacotar" a versão do software para ser distribuída e instalada, como nos casos de funcionalidades dispersas que estão sendo liberadas na mesma versão, ou haver uma necessidade de configuração específica para essa versão a ser liberada;
- **Utilizar Produto**: nesta fase, os usuários passam a utilizar a versão disponibilizada, consistindo, então, numa fase de testes de aceitação pelo usuário. A partir desses testes, podem ser remetidos problemas, sugestões, solicitações de alteração ou inclusão de requisitos, e inclusive, podem ser enviadas questões aos desenvolvedores por meio de listas de discussão ou outros meios virtuais.
- **Priorizar Melhorias**: O responsável pela versão gerada deve selecionar e priorizar, dentre as correções, sugestões e alterações apontadas pelos usuários, quais devem ser priorizadas na versão seguinte.

Os autores concluem que os projetos em software livre possuem um processo evolutivo constante e dificilmente chegam a um final definitivo, pois, mesmo que o grupo de desenvolvedores original abandone o projeto em algum momento, nada impede que outros desenvolvedores continuem o desenvolvimento em uma versão paralela. Os benefícios do modelo proposto estão na geração de código de alta qualidade, determinada pelo processo de controle de qualidade, pela validação do código-fonte, pela preocupação com a documentação de requisitos e pelos sucessivos ciclos de avaliação pelos usuários, viabilizada pela necessária disponibilização do software pelos autores.

Capítulo 4

Arquitetura Orientada a Serviços

As tecnologias para desenvolvimento de software têm se voltado para a construção de sistemas que possam estar disponíveis para qualquer pessoa que possua acesso à rede mundial de computadores, a Internet. A Web tem colaborado constantemente como alicerce no desenvolvimento de sistemas deste tipo [9].

Esta característica, por sua vez, determinou um novo requisito: a possibilidade de integração entre os sistemas, obrigando à incorporação de técnicas que permitissem a troca de informação, viabilizando a realização de computação distribuída entre esses sistemas. A Computação Distribuída visa a execução de diferentes tarefas em diversos computadores conectados por uma rede de comunicação.

Os sistemas distribuídos surgiram no momento em que diversos pacotes de programas passaram a trocar informação entre si e formaram conjuntos integrados de sistemas. Por este prisma, a Web pode ser vista como uma coleção de serviços interconectados por protocolos de comunicação. A Web utiliza serviços eletrônicos como elementos fundamentais para o desenvolvimento de aplicações distribuídas [10]. Os serviços da Web são um tipo específico de serviço eletrônico, que pode ser caracterizado como um componente aberto, auto-descritivo, padronizado, com grande interoperabilidade e acessível a outras aplicações ou serviços por meio de um endereço URL [11].

Um serviço Web também pode ser definido como sendo qualquer serviço, disponível via Internet, que se comunique com outros serviços utilizando protocolos padrão, normalmente baseados em XML, e não se limite a um determinado sistema operacional e/ou

linguagem de programação [12]. A proposta de adoção de serviços Web está fundamentada na redução dos custos ao integrar soluções diferentes, garantindo interoperabilidade entre essas aplicações com tecnologias baseada em XML, como é o caso da ferramenta estendida aqui, o Apertium.

Aproveitando toda a infra-estrutura fornecida pela Web (tecnologias, ferramentas, protocolos, entre outros) cada serviço pode representar uma funcionalidade específica, um componente integral de uma aplicação ou, até mesmo, uma aplicação completa. Com isso, os serviços Web se tornaram uma ferramenta atrativa para a construção e a concepção de sistemas distribuídos [13]. Os principais protocolos utilizados para a construção de qualquer serviço Web permitem a adição de interoperabilidade aos serviços, levando à sua caracterização como sistemas abertos [14].

Essa interoperabilidade facilita de forma significativa o desenvolvimento de aplicações distribuídas, já que problemas tais como a diversidade de hardware e software utilizada já foram considerados pelos protocolos utilizados por esses serviços [15]. Como dito anteriormente, serviços Web são bastante utilizados na integração de sistemas, capacitando os sistemas a trocarem informação de uma maneira uniforme. Toda a dinâmica de funcionamento de serviços Web é definida por uma arquitetura própria. Essa arquitetura é denominada *Service Oriented Architecture* (*SOA*) [16], que em português pode ser traduzida para Arquitetura Orientada a Serviços (AOS).

Um serviço é uma ferramenta, uma funcionalidade, uma rotina ou um método a ser implementado em uma aplicação. No contexto de SOA, são os serviços que tornam possível a recomposição total ou parcial de aplicações sem a necessidade de alterar os códigos já constituídos e devidamente testados. Além do reuso, a adoção de uma arquitetura orientada a serviços facilita a adaptabilidade de sistemas, fazendo com que eles se tornem altamente

dinâmicos, na medida em que os serviços podem ser substituídos por outros, em tempo de execução, sem impactar o funcionamento do sistema como um todo. [22]

Um serviço é acessado por meio de uma interface que inclui as especificidades do conhecimento para acessar as competências subjacentes. Não há restrições no que constitui as competências subjacentes ou a forma como o acesso é implementado pelo provedor. Então, o serviço pode ser executado como descrito na funcionalidade por um ou mais processos automáticos e/ou manuais que ele próprio invocar a outros serviços disponíveis [34].

O desenvolvimento de uma arquitetura de software orientada a serviços tem como tarefa identificar quais são os serviços participantes da solução do sistema, relacionar suas respectivas funções, entradas e saídas e gerar uma coordenação entre eles. Neste modelo conceitual da arquitetura, existem três entidades que formam o alicerce como justificativa da existência e para a compreensão da estrutura e da utilidade dos serviços. Essas entidades são os provedores de serviços, os consumidores e o repositório ou registro onde todos os serviços disponibilizados pelos provedores para serem utilizados pelos consumidores são armazenados e catalogados. Portanto, as entidades que fazem parte do funcionamento de uma aplicação de arquitetura orientada a serviços são [34]:

- **Provedor do Serviço**: é uma entidade que recebe solicitações dos e consumidores e as executa. Ele publica os serviços e contrata a interface para o registro de serviço, de modo que o consumidor possa descobrir que ele existe e ter acesso a ele;
- **Cliente do Serviço**: é constituído por uma aplicação, um módulo de software ou outro serviço que requer um serviço. O consumidor executa o serviço segundo o contrato de interface;

- **Serviço de Diretório (Registro de Serviço)**: é o facilitador para a descoberta de serviço pelo consumidor potencial. Ele contém um repositório de serviços disponíveis e os apresenta de maneira a facilitar a identificação pelos potencias interessados.

Figura 5: Funcionamento de uma Arquitetura Orientada a Serviços [36]

Aplicações orientadas a serviços são por natureza aplicações distribuídas que fazem uso dos conceitos de distribuição mencionados anteriormente. Em um sistema aberto e distribuído – sistema em que uma nova entidades pode passar a fazer parte do sistema ou deixar de fazer a qualquer momento e essas entidades podem estar distribuídas em diferentes máquinas – é necessário prever a entrada e saída de componentes no sistema de forma transparente para os seus clientes. A figura 5 representa a maneira que os serviços são publicados e utilizados segundo uma arquitetura orientada a serviços, o funcionamento ocorre as seguinte maneira: 1) Um Provedor de Serviço publica a informação de implementação de um determinado serviço no Serviço de Diretório que passa a documentar que esse provedor contem a implementação daquele serviço; 2) Um cliente acessa o serviço de diretório através de um protocolo conhecido, requisitando uma implementação de um determinado serviço; 3) O serviço de diretório acessa sua documentação de provedores para o serviço indicado e

responde dando uma identificação de um servidor/provedor que contém a implementação desse serviço juntamente com sua interface de comunicação; 4) Finalmente, o cliente acessa o serviço em questão fazendo uso correto da interface e recebe os resultados da execução [39].

4.1. Características da Orientação a Serviços

A arquitetura orientada a serviços, possui algumas características que representam todo o funcionamento do sistema que a utiliza, são elas:

- **Reaproveitamento de Código**: a principal característica do uso de orientação de serviços é o reaproveitamento de código, especialmente quando o mesmo já foi testado e incorporado em alguma outra solução consistente já em funcionamento. Esse reuso é essencialmente por meio do conceito de "caixa-preta" em que um desenvolvedor não tem acesso à forma de implementação do componente do serviço que fará parte do processo de reuso. Assim a utilização desse serviço pelo consumidor, que está reaproveitando sua funcionalidade já implementada e testada, necessita de uma descrição das interfaces, que deve ser seguida para se comunicar com o serviço por meio de um canal de comunicação pelas entradas e saídas do mesmo [34];
- **Distribuição**: outra característica de uma arquitetura orientada a serviços é que os serviços podem estar distribuídos dinamicamente, em máquinas e servidores diferentes, podendo fazer tanto distribuição de acesso à execução ou ao uso especifico de uma funcionalidade de um serviço, adequando a eficiência do tempo de execução, quanto com relação ao balanceamento de execução de varias solicitações simultâneas a um sistema [34];

- **Gerenciamento**: em sistemas orientados a serviços, o gerenciamento dos serviços é imprescindível. Com o intuito de facilitar esse controle, existem algumas ferramentas e linguagens próprias para trabalhar com serviços. Uma ferramenta de coordenação de serviços permite, por meio de repositórios, protocolos de comunicação e gerenciamento de provedores e consumidores, que um ou mais componentes se comuniquem visando um mesmo objetivo compartilhado por ambas as partes de agentes paralelos ou concorrentes [34];
- **Dinamismo e Adaptabilidade**: as aplicações orientadas a serviços são dinâmicas, já que conseguem se adaptar às mudanças de requisitos com facilidade, por meio de domínios de informação menores e mais bem definidos. Os serviços podem entrar e sair do sistema em tempo de execução, tipicamente comunicando-se com o registro de serviços por meio de mensagens de publicação ou de cancelamento de publicação. Além disso, uma boa prática indicada para trabalho com orientação a serviços consiste em que os sistemas não façam referência direta ao serviço e, sim, a uma interface de comunicação para o mesmo, que encapsula toda sua estrutura e implementação e torna possível o dinamismo e a troca de implementação interna, sem que prejudique os outros serviços já consumidores de suas funcionalidades. Essas interfaces funcionam de forma a garantirem que um componente possa ser desconectado do ambiente e substituído por outro que execute o mesmo serviço, sem provocar alterações na execução do sistema [23];
- **Sincronia**: a troca de mensagens, seu modo de transmissão e seu conteúdo são partes fundamentais dos sistemas distribuídos, pois constituem o mecanismo que os componentes possuem para comunicarem e sincronizarem suas ações [24];

- **Protocolos e Interfaces**: qualquer tipo de comunicação entre duas entidades necessita de uma linguagem entendida por ambos os lados, um protocolo. Em ambientes distribuídos, como retratados para uso de orientação a serviços, onde as partes estão conectadas por meio de uma rede de comunicação, os protocolos são ainda mais relevantes. Questões tais como desempenho e segurança estão presentes na definição da arquitetura da solução e na distribuição dos serviços, pois são por meio dos protocolos e das especificações de interfaces que os sistemas se comunicam efetivamente, havendo necessidade de garantir robustez, tanto na especificação quanto no uso das mesmas para interligar os serviços do sistema, internamente ou mesmo por meio de comunicação com serviços externos [23].

  Padrões como XML, SOAP e WSDL incluem os fundamentos de serviços de interoperabilidade. Eles asseguram que um cliente possa achar um serviço que esteja precisando e possa fazer um pedido que o cliente e o serviço entendam, independentemente de onde o cliente e o serviço residam ou da linguagem em que o cliente e/ou o serviço tenham sido codificados.

Além das características citadas, existe uma preocupação constante com a granularidade na distribuição dos serviços, pois é necessário se adequar a duas características essenciais herdadas do paradigma de sistemas orientados a objetos, a aplicação deve conter componentes de serviço com baixo acoplamento entre eles, e cada serviço deve possuir, também, alta coesão na funcionalidade a ele incumbida, de forma que possam ser utilizados por outros componentes de serviços ou até mesmo em outros projetos ou sistemas.

4.2. Definição dos Serviços

Existe uma forma padrão de definição dos serviços Web a serem implementados e/ou utilizados em um ambiente de desenvolvimento orientado a serviços, assim como de documentar essas definições.

A WSDL (Web Services Description Language) é uma linguagem de definição e documentação de serviços baseada em XML, contendo informações do que um serviço pode fazer, onde está localizado e de que forma invocá-lo [35]. Sendo baseado em XML, um documento WSDL é dividido em seções constituídas por elementos. Esse elemento contém informações a respeito de um ou mais serviços através dos seguintes dados [36]:

- **Tipos (*types*)** – contem a definição de tipos de dados usados na descrição dos serviços.
- **Mensagens (m*essages)*** – definições de tipos de dados sendo trafegados podem conter uma ou mais partes, essas partes podem ser comparadas a parâmetros de uma função ou método.
- **Tipo de porta (p*orttype*)** – um resumo da configuração das operações suportadas por um serviço.
- **Ligação (b*inding*)** – define o formato da mensagem e detalhes de protocolos para cada porta de acesso ao serviço.

Um documento WSDL pode ser dividido em documentos distintos, um que se refere à implementação de um serviço e outro que se refere à interface do serviço.

4.3. Metodologias de Desenvolvimento Orientado a Serviços

As etapas de um processo de desenvolvimento de um software que se utiliza do paradigma orientado a objetos não diferem das etapas tradicionais presentes na Engenharia de Software. A etapa de Projeto, entretanto, possui algumas características específicas para esse tipo de desenvolvimento e arquitetura de serviços.

A rápida evolução de sistemas, módulos ou mesmo serviços que se utilizem de tecnologias para a Web força com que os arquitetos e desenvolvedores a idealizarem e especificarem um projeto que resolva problemas imediatos enquanto que a arquitetura comporte uma rápida e breve evolução. Ao se resolver os problemas de arquitetura rapidamente, a capacidade evolutiva acaba sendo comprometida [40].

Sistemas que se utilizam de serviços Web tem foco na definição da estrutura do aplicativo, na aplicação de padrões e na construção de modelos permitir reuso. Um projeto feito para um software orientado a serviços deve considerar os seguintes aspectos [41]:

- **Métodos e Princípios:** Modularidade eficiente com os já mencionados princípios de orientação a objetos de alta coesão e baixo acoplamento.
- **Regras de Ouro** *(Golden Rules)***:** Sistemas para a *Web* já vêm sendo construídos há quase duas décadas e durante esse tempo os arquitetos desenvolveram um conjunto de heurísticas que podem ser reaplicadas durante as especificações de arquiteturas de novos sistemas.
- **Padrões de Projetos** (*Design Patterns*)**: A**bordagens genéricas utilizadas para resolver problemas que podem ser adaptadas para uma grande variedade de questões encontrados para o sistema em questão.

- **Modelos** *(Templates)***:** um modelo pode ser utilizado para fornecer um esqueleto para qualquer tipo de padrão de projeto que será utilizado no aplicativo, diz-se também um protótipo de arquitetura ou mesmo de definição dos serviços.

4.4. Orientação a Serviços e Software Livre

Com o objetivo de encontrar as principais características capazes de trazer o paradigma Orientado a Serviços para o ambiente de desenvolvimento de software livre, é necessário que os principais conceitos de cada uma dessas frentes de trabalho fiquem claros. O paradigma de software livre foi descrito na seção 4.1., apresentando suas principais características e vantagens de uso. Na presente seção serão resgatados os principais conceitos ligados ao software livre e aonde serão explicitados os relacionamentos deste tipo de software com a Orientação a Serviços.

Como discutido anteriormente neste texto, o software livre é um tipo de software que surge com permissão para qualquer pessoa usar, copiar e distribuir na íntegra ou com alterações, sem cobrança de licença ou qualquer outra forma de pagamento por isso. Segundo a definição da GNU [33] (Projeto GNU, lançado em 1984 para desenvolver um sistema operacional completo compatível com Unix que fosse um software livre: o sistema GNU), “O software livre é uma questão de liberdade, não de preço. Para entender esse conceito, você tem que pensar na liberdade de um discurso livre, não em uma cerveja grátis”.

A liberdade mencionada diz respeito à possibilidade de se enxergar, não apenas a funcionalidade do software, mas, principalmente, a maneira como ele foi construído, de forma que seu código fonte sempre esteja liberado para ser acessado por qualquer desenvolvedor interessado, tendo este, assim, liberdade para executar o programa, por qualquer razão,

liberdade para estudar como o programa se comporta, para adaptá-lo às suas necessidades se for preciso, podendo acessar o código fonte como condição para isso, liberdade para redistribuir cópias para auxiliar um vizinho, liberdade para melhorar o programa e distribuir essa evolução publicamente, para beneficiar a toda a comunidade envolvida. [25]

A liberdade de executar o programa significa a liberdade para qualquer pessoa ou organização utilizá-lo em qualquer tipo de sistema de Computação, para qualquer tipo de trabalho, sem ser obrigada a comunicar sobre isso ao autor ou qualquer outra entidade específica. A liberdade de redistribuir cópias deve incluir formas binárias ou executáveis do programa, bem como o código fonte, para ambas as versões, a original e a modificada. Com o objetivo de tornar a liberdade de fazer modificações e publicar versões melhoradas viável, é necessário ter acesso ao código-fonte do programa. [25]

É possível traçar um paralelo entre estas características associadas à liberdade de uso e reuso do software livre, e as características inicialmente apontadas para o paradigma de Orientação a Serviços. No paradigma de Orientação a Serviços, uma vez que um serviço está desenvolvido e devidamente testado e tornado disponível no repositório, ele passa a poder ser utilizado livremente por qualquer pessoa ou projeto que reuse sua funcionalidade, poupando esforço e tempo de implementação.

Se unirmos os pilares de ambos os conceitos mencionados, temos que um serviço pode ser desenvolvido com seu código aberto (premissa do software livre) e implementado contendo suas interfaces de uso bem definidas (princípios do paradigma Orientado a Serviços). Assim, qualquer desenvolvedor poderia, além de se limitar a usar suas funcionalidades, alterar a forma de implementação a fim de melhorar o serviço ou alterá-lo para outras necessidades e usos, recolocando novamente o serviço alterado no repositório. Aqui ocorre uma mudança no conceito inicialmente descrito para um serviço, pois sua

implementação deixaria de ser uma “caixa-preta”, em que o consumidor do serviço desconhece o código que gera aquela funcionalidade, passando para um conceito de “caixa-branca”, interesse chave entre os ideais do software livre, pois o código passa a ser visível, podendo ser estudado e alterado. Deve haver para isso uma restrição de manter sempre compatíveis as entradas e saídas, ou seja, a interface de comunicação desse serviço com outras entidades, de forma a evitar a perda da compatibilidade desse serviço evoluído com todos os processos que o utilizam.

Uma forma importante para modificar um programa está disponível a partir da fusão (*merge*) de sub-rotinas, de módulos e de métodos. Esse conceito trazido do software livre vai ao encontro do conceito do reuso de serviços pelo paradigma de Orientação a Serviços, pois se for definida uma granularidade de serviços em nível de pequenas funcionalidades, métodos ou rotinas, e estas forem fundidas originando novos serviços, encontra-se a possibilidade de geração de novos serviços produzidos pela junção de outros dois menores, prática característica do software livre.

Através do exposto, é possível verificar pontos convergentes entre as vertentes de orientação a serviços e software livre, justificando o interesse em atender o problema de hipótese deste trabalho, através de uma metodologia de desenvolvimento de sistemas que trabalhem com ambos os conceitos.

Capítulo 5

Metodologia Proposta

Com o levantamento de diversos métodos e processos de desenvolvimento e o conhecimento sobre o contexto de máquinas de tradução e de arquiteturas orientadas a serviços, será descrita no presente capítulo, a metodologia composta e o resultado obtido no uso dessa metodologia para o problema considerado, sendo este uma extensão do aplicativo Apertium para aceitar bases de conhecimento oriundas de outros aplicativos, contribuindo para uso e a complementação de outros trabalhos de pesquisa paralelos.

5.1. Metodologia de Desenvolvimento

5.1.1. Etapas do Processo de Software

As atividades consideradas essenciais para a construção de um software livre, utilizando uma arquitetura orientada a serviços, podem ser resumidas em:

- **Elicitação de Requisitos**: É sugerido aos projetos realizar um processo de Engenharia de Requisitos que contempla toda uma metodologia de trabalho da Engenharia de Software, normalmente com a participação do usuário final do sistema, por ser um bom conhecedor do domínio do problema que o software deve solucionar. [20]

  Um requisito pode ser definido como:

  1. uma funcionalidade no nível do usuário; por exemplo, um corretor de gramática e ortografia;

2. uma propriedade muito geral do sistema; por exemplo, o sigilo de informações não autorizadas;
3. uma restrição específica no sistema; por exemplo, o tempo de varredura de um sensor;
4. uma restrição no desenvolvimento do sistema; por exemplo: a linguagem que deverá ser utilizada para o desenvolvimento do sistema.

Existem dois tipos de requisitos: funcionais e não-funcionais. Os requisitos funcionais referem-se aos requisitos que estão relacionados com a maneira com que o sistema deve operar, onde se especificam as entradas e saídas do sistema e o relacionamento comportamental entre elas, assim como a iteração com o usuário. Os requisitos não-funcionais são aqueles que não estão especificamente relacionados com a funcionalidade do sistema. Eles impõem restrições no produto a ser desenvolvido e/ou no processo de desenvolvimento do sistema assim como especificam restrições externas a serem atendidas. Referem-se a questões tais como: segurança, confiabilidade, usabilidade, desempenho, entre outras.

Esta etapa de elicitação de requisitos consiste em entender o objetivo do software a ser desenvolvido, descobrindo e detalhando quais funcionalidades estarão contempladas nele, essas farão parte dos requisitos funcionais do sistema. Podem, ainda, ser levantados requisitos relacionados à forma em que o software deve ser desenvolvido, assim como restrições de banda de comunicação, de hardware de máquinas, de tempo de resposta dos processos, de espaço de armazenamento em disco e em memória da aplicação, entre outros, constituindo uma leva de requisitos não-funcionais. Esta atividade foi apresentada e sugerida por [26] e [27] em seus trabalhos de análise e definição de processos de software, para software livre.

- **Definição da Arquitetura e Desenho da Solução**: estas são atividades de cunho técnico, de detalhamento iterativo de uma solução computacional para atender aos requisitos (especialmente aos não-funcionais). Essas etapas na Engenharia de Software são conhecidas como Análise e Projeto. Em níveis mais superficiais, são definidas as entradas e saídas do software e as entidades (atores) com as quais ele se relaciona, podendo estes ser humanos ou outros sistemas. Progressivamente, a etapa de Projeto realiza um refinamento do problema a ser desenvolvido para o software em partes menores, constituindo módulos e funções separadas. Além disso, determina a arquitetura a ser utilizada pela aplicação, com suas características e limitações, gerando um modelo ou um desenho da estrutura da solução, contemplando os servidores de aplicação, o banco de dados, os protocolos de comunicação, entre outros.

  A definição da arquitetura deve ser feita e revisada por pessoas experientes, especialmente com conhecimentos específicos e sobre o ambiente do projeto. A partir disso, uma documentação bem detalhada deve ser gerada a fim de sanar ao máximo as dúvidas e indefinições que ocorram na fase de codificação. Esse documento de arquitetura, ou documento de projeto, deve estar igualmente versionado e visível para todos os envolvidos no processo, especialmente na fase de codificação.

  No contexto de Orientação a Serviços, como descrito no capítulo 4, essa etapa é a que define quais serão os serviços disponibilizados pela aplicação, principalmente em qual granularidade eles serão propostos. É importante salientar que em um ambiente assim, a definição da arquitetura do sistema é fundamental para estruturar qual serviço cada servidor poderá executar, juntamente com uma

definição de segurança de acesso e encapsulamento desta informação. Esta atividade foi apresentada e sugerida para projeto Apache (seção 3.3.) e para [26] em seu trabalho de análise e definição de processos de software, para software livre.

- **Codificação**: esta atividade consiste na elaboração do código fonte, ocorrendo o desenvolvimento prático da solução, implementando as funcionalidades descritas nos requisitos funcionais e adequando a solução ao modelo definido na etapa de Análise e Desenho da solução. Esta atividade normalmente ocorre progressivamente, onde novas funcionalidades são criadas e integradas a funcionalidades já existentes. No contexto de Orientação a Serviços, a codificação ocorre no nível dos serviços identificados, respeitando as configurações planejadas e a decorrente arquiteta definida na fase de Projeto. Esta atividade foi apresentada e sugerida por todos os autores mencionados nesta dissertação, por ser a etapa que cria o software propriamente dito.
- **Testes**: esta atividade ocorre dividida em duas partes. A primeira ocorre em paralelo à etapa de codificação, onde o desenvolvedor coloca o seu código para ser testado separadamente do restante do sistema, representado um teste unitário, por testar uma unidade de processamento do sistema isoladamente, ou seja, um serviço que deve funcionar de maneira independente, coesa e com baixo acoplamento em relação a outras funcionalidades. A segunda etapa testa agrupamentos de funcionalidades integradas e em funcionamento conjunto e dependente, validando aspectos funcionais do sistema, incluindo, também, os requisitos não-funcionais tais com desempenho e limitações exigidas, representando testes do sistema como um todo. Como o ambiente considerado é de software livre, não existe o

compromisso de realização das tarefas pelas pessoas, então os próprios desenvolvedores devem realizar um teste cruzado de funcionalidades, o que consiste em que um desenvolvedor realize testes de serviços de um outro e vice-versa no ambiente de desenvolvimento de origem. Esta atividade foi apresentada e sugerida para o projeto Linux (seção 3.2.), projeto Apache (seção 3.3.) e também para [27] em seu trabalho de análise e definição de processos de software, para software livre.

- **Homologação ou Lançamento**: uma vez que o sistema (ou mesmo um módulo ou um conjunto de serviços) estiver inteiramente desenvolvido e testado, tendo esgotada a busca de inconsistências, deve ser feita uma liberação para a execução do sistema pelos usuários potenciais, que testarão suas funcionalidades, apresentando retorno sobre possíveis problemas relativos às funcionalidades ou à usabilidade do sistema. Esta atividade foi apresentada e sugerida para o projeto Linux (seção 3.2.) e também para [26] e [27] em seus trabalhos de análise e definição de processos de software, para software livre. O lançamento do software implementado deve ser disponibilizado na Internet, através de uma GPL (*GNU Public License*). Criada pela *Free Software Foundation*, a GPL estabelece regras rígidas para a cópia, redistribuição e alteração do software. Fundamentalmente, ela quer garantir que qualquer software livre licenciado por ela mantenha sempre esta condição, isto é, se o software for alterado e redistribuído, essa alteração passa a ser software livre também e deve estar disponível sob os mesmos termos. Para aplicar as regras exigidas pela GPL, é preciso:

    1. adicionar uma cópia do arquivo LICENSE junto ao código-fonte do programa;

2. colocar um arquivo README em um diretório *about* (ou sobre em português);
3. adicionar em cada arquivo do código-fonte (como cabeçalho, dentro de um comentário) o seguinte trecho.

```
<uma linha com o nome do programa e o que faz>
Copyright (C) 2003-2004 <autor> <email>
Author(s): <o(s) vosso(s) nome(s)>

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307  USA
```

Figura 6: Modelo de Cabeçalho de Código-Fonte [30]

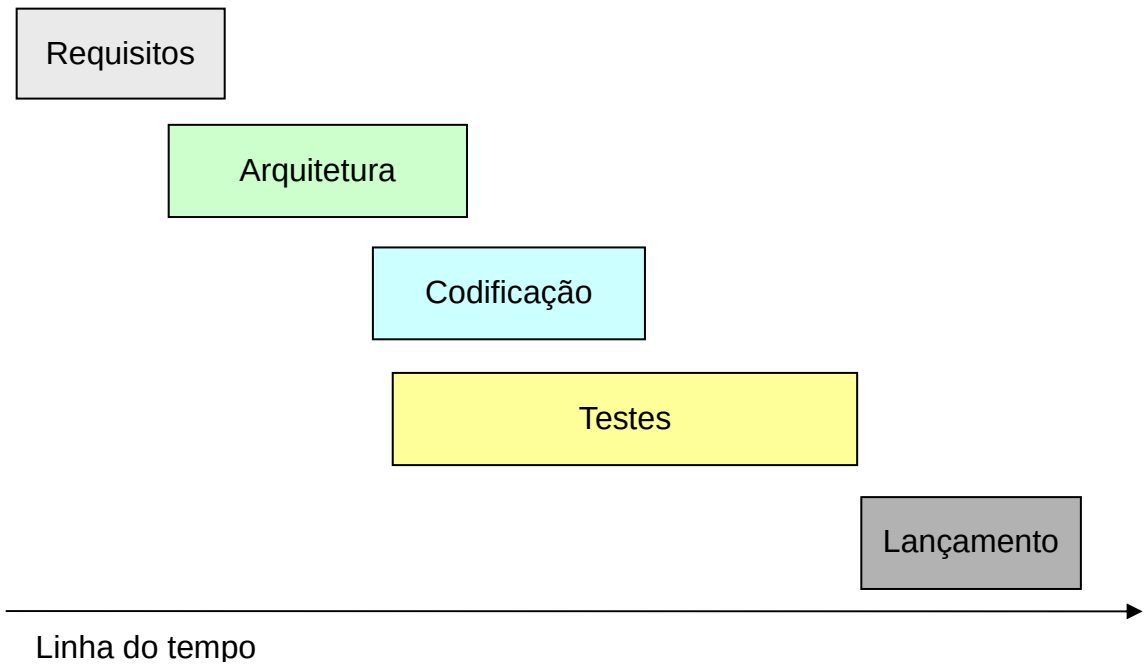

Figura 7: Modelo de desenvolvimento proposto para software livre orientado a serviços

A figura 6 apresenta uma cronologia das atividades, de acordo com o andamento das etapas sugeridas, cujo desenvolvimento se adeque à metodologia aqui descrita. Nem sempre as atividades ocorrem de maneira síncrona e seqüencial, pois, dentro de um sistema com uma vasta gama de funcionalidades e requisitos, atividades independentes podem estar em diferentes etapas no fluxo do processo.

Este fluxo de processo, apesar de ser direcionado para software livre, pode ser utilizado para o desenvolvimento de sistemas de qualquer natureza. No entanto, trabalhando no contexto de software livre, algumas atividades devem ser executadas de uma maneira particular. Dadas as características do desenvolvimento em software livre estudadas até aqui, devem ser incluídos alguns artefatos voltados para esse contexto, especialmente na evolução do projeto e na atribuição das responsabilidades.

5.1.2. Características do Processo de Software

Para o desenvolvimento de software livre, várias características foram apresentadas no capítulo 3, que contém a resenha literária sobre metodologias de desenvolvimento em software livre. Estas características devem ser consideradas no uso da metodologia que está sendo proposta nesta seção. Durante o processo de desenvolvimento do projeto, quando de projetos com muitas pessoas, especialmente no caso de estarem dispersas geograficamente, algumas atividades de controle e gerenciamento também devem ser realizadas. Assim, as características e atividades de gerenciamento consideradas relevantes pelo autor da presente dissertação são:

- **Equipe de Colaboradores**: os colaboradores envolvidos no projeto no software, que terão participação nas atividades previstas pela metodologia no fluxo de desenvolvimento do software, podem ser definidos como:

1. Usuários não-participantes: sem participação no desenvolvimento do software, ajudam apenas no uso e na validação do sistema, posteriormente ao seu lançamento na fase de homologação.
2. Usuários participantes: atuam ativamente no desenvolvimento do projeto, ajudando na definição dos objetivos do software e no levantamento dos requisitos, sendo também responsáveis por boa parte dos testes do sistema, provendo retorno para a equipe de desenvolvimento com relação às inconsistências e melhorias.
3. Desenvolvedores esporádicos: são usuários com conhecimento em Computação, especialmente em Programação, que desenvolvem pequenos serviços mais independentes e/ou realizam correções de inconsistências encontradas nos testes ou outras alterações de pequeno impacto no escopo geral do projeto.
4. Desenvolvedores ativos: normalmente são os próprios idealizadores do projeto, que têm responsabilidade por grandes módulos e conjuntos de funcionalidades, em todas as definições dos serviços e do ambiente que será necessário para executar o sistema. São estes desenvolvedores que devem planejar a arquitetura do sistema e viabilizar a implementação dos requisitos não-funcionais levantados na etapa de elicitação de requisitos.
5. Equipe de controle de qualidade: estes participantes não devem se envolver com nenhuma atividade do processo de desenvolvimento do sistema, fazendo apenas um papel posterior de revisores e auditores dos artefatos e dos processos gerados. No contexto de software livre, em não havendo compromisso de realização das tarefas nem das datas, é interessante que

haja um trabalho cruzado de controle de qualidade, onde desenvolvedores de um projeto participem da equipe de *Software Quality Assurance* (SQA) de um outro projeto, e os desenvolvedores desse outro projeto façam parte da equipe de SQA do primeiro.

- **Coordenação:** Existe uma tarefa fundamental, especialmente em projetos de software livre, denominada "gerência de alocação", que consiste em atribuir e administrar recursos – ambiente, máquinas, configurações, pessoas – para as diferentes atividades do projeto, com o objetivo de otimizar a produtividade e a adequação aos requisitos determinados para o sistema. Considerando projetos que tenham, também, necessidades de controle de prazo e de liberação de versões, um cronograma se faz necessário para que sejam controlados o andamento das atividades e as metas para atingir os prazos esperados de conclusão das diversas tarefas e do projeto como um todo.

  Existem outras atividades de controle de possíveis regras e políticas estabelecidas para o desenvolvimento do projeto, principalmente para grandes escopos e grandes equipes, tais como: controle de versões do software, uso de ferramentas de comunicação, administração dos *bugs* detectados pelos testes ou pelos usuários do sistema que devem ser corrigidos em próximas versões, controle de qualidade ao longo do processo, documentação, entre outras.

- **Controle de Qualidade**: embora visto na revisão de metodologias de desenvolvimento de software livre do capitulo 3, que nem sempre há uma preocupação com a qualidade do software que está sendo desenvolvido, a metodologia aqui proposta estabelece como principio básico a adoção dessa prática, como visto no projeto Apache, que existe um sub-projeto específico para

trabalhar com o controle de qualidade do código que está sendo desenvolvido, o *Apache HTTP Test Project*, uma equipe focada em validar a forma de implementação, a lógica dos programas, as definições e padrões estabelecidos, entre outras coisas. Nas práticas modernas da Engenharia de Software, independentemente de se trabalhar com software proprietário ou software livre, um processo de validação, tanto dos artefatos que estão sendo gerados quanto dos processos que os geram, é realizado como garantia da qualidade, o já mencionado SQA, que contempla uma série de procedimentos e políticas estabelecidas para supervisionar a qualidade do que estiver sendo avaliado, o que pode ser artefatos, modelagens e fontes, além de adequação dos processos. Essa validação precisa ocorrer em todas as etapas previstas pelo fluxo de desenvolvimento de software.

- **Comunicação**: as pessoas envolvidas no projeto devem se comunicar de maneira simplificada, pois é chave do desenvolvimento de qualquer sistema em software livre, especialmente pela característica de dispersão dos membros da equipe. Desta forma, deve ocorrer na Internet, por meio de ferramentas que se encontram amplamente disponíveis gratuitamente, tais como correio eletrônico, paginas Web, fóruns e listas de discussão, entre outros, em conjunto com as ferramentas previstas para o controle e gerenciamento do software.
- **Documentação**: toda a documentação gerada para o projeto, principalmente os arquivos do código-fonte do software, deve ser protegida e devidamente versionada por uma ferramenta adequada. Uma vez que o trabalho será desenvolvido em paralelo e até com os desenvolvedores espalhados geograficamente, é necessário ter um repositório único da solução em algum local seguro, que conte com programas de controle de versões dos arquivos e

gerenciamento de *backups*, como o CVS, apresentado na revisão do projeto Apache. O CVS permite que um desenvolvedor trabalhe isoladamente, em uma base local, em uma versão dos arquivos que está bloqueado para ele, sem necessitar estar conectado a outro local e sem conflito com o trabalho de outros desenvolvedores. Uma vez que o código-fonte fica protegido e versionado, em qualquer momento antigas versões dos arquivos fonte podem ser analisadas, acionando o nome dos autores que fica sinalizado junto a cada versão. Quando os arquivos estão estáveis e se desejar congelar uma versão, com o intuito de ela ser testada, lançada ou apenas para ter um controle de um conjunto de arquivos agrupados, a funcionalidade de gerar uma etiqueta (*label*) de identificação dessa versão pode ser acionada.

A documentação precisa ocorrer de uma maneira eficiente para atingir a todos os membros da equipe de maneira satisfatória. Na revisão das metodologias de desenvolvimento do capitulo 3, foi visto que a documentação em sistemas de software livre é praticamente a única forma de descrição da estrutura do código, das decisões de arquitetura e do mapeamento dos requisitos. Esta documentação consiste normalmente em descrições textuais ou gráficas, em formato padrão e com conteúdo passível de compreensão por todos os interessados. Na Engenharia de Software, são descritos diferentes formatos de documentação para cada etapa do desenvolvimento de software, sendo cada formato e artefato adequado para um contexto (domínio da aplicação e perfil das pessoas envolvidas).

- **Controle de correções**: esta atividade deve ser realizada por ferramentas específicas para esse fim, como descrito no modelo de desenvolvimento do Apache, que usa o Bugzilla para realizar esse controle de maneira satisfatória.

5.2. Estudo de Caso

A metodologia descrita na seção anterior foi aplicada ao desenvolvimento de uma extensão para a máquina de tradução Apertium. O processo inclui a implementação da importação de uma base de conhecimento proveniente de outros aplicativos para a realização do processo de tradução considerando essa nova base de conhecimento, também inclui interfaces que permitam que qualquer uma das etapas de tradução, do modelo do Apertium, seja acionada ou substituída por outras ferramentas.

Nesta seção serão apresentados os artefatos obtidos nessa atividade, a fim de ilustrar o processo de desenvolvimento da solução do problema estudado.

5.2.1. Elicitação de requisitos

Os requisitos definem os serviços que o sistema deverá oferecer, e o conjunto deles determina a operação do sistema, conforme descrito na seção 5.1.1. Os requisitos levantados para o desenvolvimento do software descrito deveriam estar inseridos em um documento formal que seja versionado e visível para todos os envolvidos no processo. Futuramente, dependendo da definição do padrão de documentação do projeto, esses requisitos podem fazer parte da documentação do projeto. Para realização desta etapa, definindo os requisitos deste sistema, foi considerado, junto a trabalhos de pesquisa em paralelo, qual era a necessidade de extensão do software em questão, o Apertium. Uma extração do documento de requisitos no caso estudado conteria:

Requisitos

Requisitos Funcionais

Os requisitos funcionais encontrados para o ciclo atual do projeto são:

ID	Requisito	Descrição
[RF01]	Realizar importação de arquivo contendo o mapeamento das regras de tradução entre um par de línguas	Permitir que um serviço seja substituído por um serviço Web de uma ferramenta cliente, de forma que seja informado qual o arquivo a ser importado para o Apertium, em qual caminho deve ser colocado esse arquivo nos diretórios do Apertium e qual o par de línguas a que ele se refere, retornando sucesso ou insucesso da operação.
[RF02]	Consultar os arquivos de mapeamento de regras de tradução entre pares de línguas existentes	Permitir que um serviço seja executado por um serviço Web de uma ferramenta cliente, de forma que seja solicitada uma lista de arquivos de mapeamento de regras de tradução entre pares de línguas, presentes nos diretórios do Apertium, retornando uma lista descrevendo esses arquivos.
[RF03]	Executar o módulo Desformatador	Permitir que um serviço seja executado por um serviço Web de uma ferramenta cliente, de forma que seja informado um conteúdo a ser processado pela etapa de tradução Desformatador do Apertium, retornando o conteúdo resultante desse processamento.
[RF04]	Executar o módulo Analisador Morfológico	Permitir que um serviço seja executado por um serviço Web de uma ferramenta cliente, de forma que seja informado um conteúdo a ser processado pela etapa de tradução Analisador Morfológico do Apertium, retornando o conteúdo resultante desse processamento.
[RF05]	Executar o módulo Desambiguador Léxico	Permitir que um serviço seja executado por um serviço Web de uma ferramenta cliente, de forma que seja informado um conteúdo a ser processado pela etapa de tradução Desambiguador Léxico do Apertium, retornando o conteúdo resultante desse processamento.

[RF06]	Executar o módulo Transferência Estrutural	Permitir que um serviço seja executado por um serviço Web de uma ferramenta cliente, de forma que seja informado um conteúdo a ser processado pela etapa de tradução Transferência Estrutural do Apertium, retornando o conteúdo resultante desse processamento.
[RF07]	Executar o módulo Transferência Léxica	Permitir que um serviço seja executado por um serviço Web de uma ferramenta cliente, de forma que seja informado um conteúdo a ser processado pela etapa de tradução Transaferência Léxica do Apertium, retornando o conteúdo resultante desse processamento.
[RF08]	Executar o módulo Gerador Morfológico	Permitir que um serviço seja executado por um serviço Web de uma ferramenta cliente, de forma que seja informado um conteúdo a ser processado pela etapa de tradução Gerador Morfológico do Apertium, retornando o conteúdo resultante desse processamento.
[RF09]	Executar o módulo Pós-Gerador	Permitir que um serviço seja executado por um serviço Web de uma ferramenta cliente, de forma que seja informado um conteúdo a ser processado pela etapa de tradução Pós-Gerador do Apertium, retornando o conteúdo resultante desse processamento.
[RF10]	Executar o módulo Reformatador	Permitir que um serviço seja executado por um serviço Web de uma ferramenta cliente, de forma que seja informado um conteúdo a ser processado pela etapa de tradução Reformatador do Apertium, retornando o conteúdo resultante desse processamento.
[RF11]	Executar uma tradução de um texto entre um par de línguas	Permitir que um serviço seja executado por um serviço Web de uma ferramenta cliente, de forma que seja informado um texto em uma língua de origem a ser traduzido e o par de línguas a ser utilizado pela tradução, retornando o mesmo texto traduzido para a língua destino.

Requisitos Não-Funcionais

O único requisito não-funcional encontrado para o caso estudado é:

ID	Requisito	Descrição
[RNF01]	Sistema Operacional Linux	Os serviços implementados devem utilizar o Sistema Operacional Linux Ubuntu versão 8.0.4., para se adequarem à versão padrão da ferramenta Apertium que já vem embutida nessa versão.

5.2.2. Arquitetura

A definição e documentação da arquitetura é a segunda etapa prevista na metodologia que está sendo proposta, conforme descrito na seção 5.1.1. A partir das necessidades descritas pelos requisitos, do estudo e testes com a ferramenta Apertium que está sendo estendida, de conhecimento e pesquisa sobre arquitetura orientada a serviços, no que se refere a *frameworks*, padrões, entre outros, foi definida a arquitetura e os aspectos necessários para que seja realizada a implementação desse sistema. Assim, em uma extração do documento de projeto que conteria estas definições seria:

Introdução

Este documento tem por objetivo descrever a arquitetura interna e o projeto de desenvolvimento do software extensão da ferramenta Apertium. Essa extensão compreende interfaces de comunicação desse software com qualquer outro que queira integrá-lo por meio do uso de serviços Web. O projeto será desenvolvido como um sistema Web, ou seja, ao fim do processo serão disponibilizados serviços da grande rede que implementam as funcionalidades previstas.

A arquitetura aqui descrita se baseia na análise feita do Documento de Requisitos do projeto, a partir de estudos de viabilidade e de tecnologia a ser utilizada. Os requisitos definem os serviços que o sistema deverá oferecer, e o conjunto deles determina a operação do sistema.

Ao fim do documento espera-se chegar a um nível de abstração suficiente para que a equipe de implementação, independentemente de trabalhar de forma dispersa ou presencial, tenha condições de codificar, testar e entregar o produto.

2. Metodologia

O processo utiliza uma metodologia identificada para o desenvolvimento de software livre orientado a serviços. Nessa metodologia, a produção de artefatos ocorre de maneira seqüencial, com paralelismo apenas em atividades não interligadas, que é uma produção no qual o desenvolvimento é comparável ao fluir de águas, constantemente e para a frente, por meio das fases de requisitos, arquitetura, implementação, testes, e liberação.

3. Tecnologia

A tecnologia a ser empregada no desenvolvimento deste software se baseia no paradigma SOA (*Service Oriented Architecture*), ou seja, de uma definição de serviços a serem disponibilizados no formato de serviços Web.

A arquitetura orientada a serviço introduz uma camada lógica (serviços) dentro da plataforma de computação distribuída. Essa camada lógica de integração pode ser vista como sendo uma interface de comunicação entre a aplicação (Apertium), permitindo-as interagirem oferecendo e consumindo serviços uma da outra.

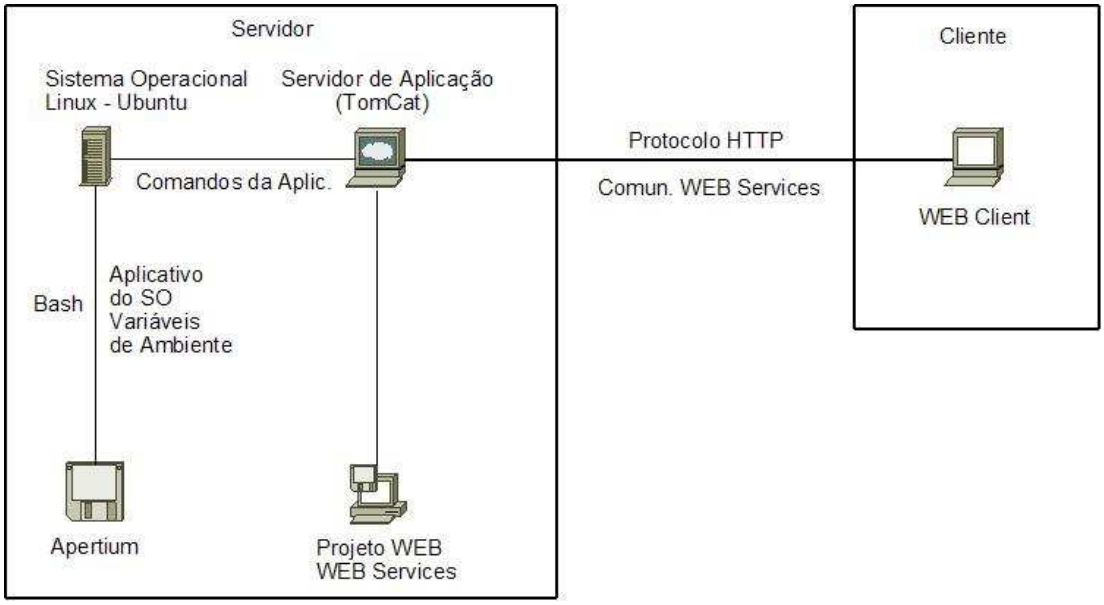

Figura 8: Arquitetura do sistema

A figura 6 representa, de maneira geral, o funcionamento da arquitetura prevista para este projeto. Os diversos aspectos desta arquitetura são apresentados a seguir:

- Linguagem Java, JDK 1.5.0 10.
- Servidor de Aplicação Web Apache 6 (Apache HTTP Server). Disponibiliza aplicativos da Web utilizando o protocolo HTTP ou HTTPS e serve de *container* para a utilização do *framework* Axis.
- A implementação dos serviços que serão disponibilizados pela aplicação deverá conter as funcionalidades do framework Axis da plataforma Java (Apache Axis 1.4).
- Apache Axis é uma implementação do SOAP submetido ao W3C (*World Wide Web Consortium)* que desenvolve tecnologias, softwares e ferramentas para Web. O *framework* Axis é uma ferramenta que auxilia na criação de um Web service. SOAP (*Simple Object Access Protocol*) é um protocolo para a troca de informações de maneira estruturada em um ambiente distribuído. É um protocolo baseado em XML que consiste em três partes: um envelope que define um quadro para descrever o que está em uma mensagem e a forma de processá-lo; um conjunto de codificação de regras para os casos de aplicação; e uma convenção para representar chamadas de procedimento remoto e respostas.

- Serviços Web em Java usando o Framework, para trabalhar com SOA, assim é criada uma camada de métodos (serviços).
- O funcionamento do framework AXIS consiste em transformar os métodos públicos em serviços, criando um arquivo de definição de serviço WSDL, um arquivo em formato XML, que segue um padrão SOA. WSDL (*Service Web Definition Language*) define regras para descrever os serviços, e sua interface detalhada, com definição de atributos e regras de utilização, de forma padronizada. As definições independem de plataforma ou de linguagem de programação. Esses arquivos ficam transparentes na arquitetura, pois o AXIS encapsula e cria o arquivo de definições, não sendo necessário a criação manual do arquivo ou o conhecimento das regras e padrões do WSDL, facilitando o desenvolvimento.
- O código Java, interno à implementação dos serviços, se comunica com o Apertium via comandos para o SO Linux, onde o aplicativo tradutor e os pares de línguas estão instalados. A aplicação Java implementa as interfaces de acesso para os serviços Web.
- O Sistema Operacional deve ser o Linux Ubuntu, útil para uso em aplicações servidoras. Deve ser utilizada a versão 8.04. A escolha dessa distribuição deu-se pela facilidade de utilização e pela simplificação da integração do aplicativo Apertium, cuja distribuição vem disponibilizada pela instalação do SO.

5.2.3.Codificação

A fase de Codificação deve ser gerenciada por um dos idealizadores do projeto. Dessa forma, no caso em estudo, por se tratar de uma equipe muito pequena, formada por uma pessoa e em poucos momentos por duas pessoas, as solicitações de implementação foram feitas e gerencias diretamente por correio eletrônico, sem a necessidade de criação de um cronograma formal ou de um grupo de discussão, conforme mencionado na seção 5.1.2. A metodologia apresenta flexibilidade quanto à necessidade ou não de formalidades como essa.

Como controle de versão foi utilizado um projeto no repositório da ferramenta CVS, sincronizando e garantindo a segurança dos arquivos, conforme o funcionamento da ferramenta, explicado na seção 3.3.

5.2.4. Testes

Novamente, por se tratar de uma equipe muito pequena, os testes foram realizados pelo próprio desenvolvedor, no caso o autor da presente dissertação, tratando rapidamente os erros encontrados após o término do desenvolvimento. Para a evolução deste projeto, será necessária a aplicação, prevista na metodologia, de um software de controle de erros e alterações, como o Bugzilla, explicado na seção 3.3.

5.2.5. Lançamento

O software implementado será disponibilizado na rede, por meio de um GPL (*GNU Public License*), conforme descrito para essa etapa na seção 5.1.1.

5.2.6. Características e Atividades do Projeto

O projeto de implementação do software para o estudo de caso contou com algumas das características citadas na seção 5.1.2. Como:

- **Equipe de Colaboradores**: o projeto contou com poucos colaboradores. Pelo escopo de atuação houve apenas 3 participações: um desenvolvedor esporádico que auxiliou apenas na definição da arquitetura a ser utilizada, um desenvolvedor ativo, o autor do presente trabalho, que trabalhou em todas as etapas e foi responsável pela idealização e execução do projeto, e um usuário participante, o

qual ajudou a definir os requisitos para condizer com suas necessidades em um projeto de pesquisa paralelo.

- **Coordenação:** tarefa pouco exercida pelo escopo reduzido e trabalho em equipe praticamente inexistente no projeto. O autor do presente atuou como coordenador ao executar e direcionar o trabalho de acordo com as etapas definidas na metodologia.
- **Controle de Qualidade**: tarefa apresentada como primordial pela definição da metodologia. Foi executada pelo próprio desenvolvedor principal do projeto, por não haver uma equipe formada para este fim.
- **Comunicação**: como dito, a equipe de colaboradores do projeto foi pequena e a comunicação foi feita diretamente por correio eletrônico quando preciso.
- **Documentação**: toda a documentação gerada para o projeto, principalmente os arquivos do código-fonte do software, documentos de especificação de requisitos e de projeto, foi devidamente versionada pela ferramenta CVS.
- **Controle de correções**: esta atividade ainda não foi executada no projeto, sendo prevista e necessária na evolução do mesmo.

Essas características e atividades apresentadas reforçam o uso da metodologia, a qual ajudou no processo de desenvolvimento do software do estudo de caso.

Capítulo 6

Conclusões e Trabalhos Futuros

Tendo como hipótese um sistema em software livre com uma arquitetura orientada a serviços - o software Apertium, uma máquina de tradução objeto de estudo do presente trabalho e de outros trabalhos paralelos - e uma necessidade específica de desenvolver uma extensão para ele, o autor do presente trabalho fez uma revisão de literatura, incluindo trabalhos relacionados e conhecimento no âmago da comunidade de software livre, a título de instrumentação teórica para a construção de uma metodologia de desenvolvimento que atendesse o caso de hipótese e apresentasse uma sugestão de trabalho para casos como esse, de desenvolvimento de software livre com arquitetura orientada a serviços.

Com base no levantamento e na análise de diferentes metodologias de desenvolvimento conhecidas e utilizadas na comunidade de software livre, assim como de alguns trabalhos que discutem o processo de desenvolvimento de software nesse contexto, para variados casos, identificando as principais características, vantagens e desvantagens, assim como os artefatos elaborados, foi composta uma metodologia de desenvolvimento de software livre e, para ilustrar seu uso, foi utilizado o ser arcabouço a fim de desenvolver um software que fosse uma extensão do Apertium.

Desta forma, pretendeu-se por meio do presente trabalho, criar uma proposta de metodologia de desenvolvimento em software livre, com foco em orientação a serviços, que contempla as principais necessidades descritas na Engenharia de Software e atenda, na prática, os projetos de desenvolvimento existentes na comunidade de software livre. Isto se traduz na adequação de uma equipe de desenvolvimento geograficamente dispersa, na

necessidade de controle de versões, erros e alterações, em canais de comunicação viabilizados pela Internet e, principalmente, a uma etapa de pré-implementação para documentação do projeto, visando definir a arquitetura e funcionamento do projeto como um todo, moldando a implementação e a definição dos serviços que serão elaborados.

O estudo de caso do presente trabalho (que consistiu no desenvolvimento de novos serviços a serem integrados no Apertium) mostrou que a construção de software livre orientado a serviços, atendendo aos princípios próprios da comunidade em questão, é passível de realização por meio da solução apresentada.

Como trabalhos futuros pode ser citada uma utilização da metodologia proposta em um projeto de sistemas orientados a serviços com um escopo maior, fugindo do ambiente de uma máquina de tradução, para consolidar as etapas identificadas e ratificar sua importância no processo de desenvolvimento de um software livre com essas características.

Um outro trabalho futuro e complementar seria efetuar analises e fomento de outros domínios de sistemas, que não utilizem arquiteturas especificas como feito neste trabalho para orientação a serviços, e tentar formatar metodologias de desenvolvimento especificas para cada domínio ou até mesmo uma metodologia bem genérica que atenda qualquer tipo de sistema em software livre.

Por último, como um trabalho futuro interessante na área de pesquisa em software livre, descobrir como incentivar a escrita de software livre em domínios ainda não atingidos atualmente. É importante prover as metodologias e ferramentas de apoio que possam auxiliar os desenvolvedores, gestores e usuários em projetos de desenvolvimento em software livre, dando segurança, reforçando seus benefícios e incentivando seu uso, trazendo inclusive mais países e segmentos da sociedade que ainda não têm presença relevante neste universo.

Referências Bibliográficas

[1] ALVES, J., CAMPOS, P., BRITO, P. *O Futuro da Internet – Estado da Arte e Tendências de Evolução*, Lisboa: Centro Atlântico, (1999).

[2] GOMES, F., PARDO, T., Estudo e aprimoramento do sistema Apertium de tradução automática entre português e espanhol, Instituto de Ciências Matemáticas e de Computação (ICMC), USP/São Carlos.

[3] NIRENBURG, S., S. BEALE, C. DOMASHNEV., *A Full-Text Experiment in Example-Based Machine Translation Proceedings of the International Conference on New Methods in Language Processing*, Manchester, England, (1994).

[4] ARMENTANO-OLLER, C. et al., *Apertium, Una plataforma de código abierto para el desarrollo de sistemas de traducción automática*, Grup Transducens, Departament de Llenguatges i Sistemes Informàtics, Universitat d'Alacant, E-03071 Alacant

[5] FORCADA, M. et al, *Documentation of the Open-Source Shallow-Transfer Machine Translation Platform Apertium,* Departament de Llenguatges i Sistemes Inform`atics Universitat d'Alacant, (2008).

[6] NIRENBURG, S. "Knowledge and Choices in Machine Translation". *Machine Translation. Org. Sergei Nirenburg. Cambridge, Cambridge University Press*, (1987).

[7] SLOCUM, J. *A Survey of Machine Translation: Its History, Current Status, and Future Prospects*. *Machine Translation Systems, Org. Jonathan Slocum. Cambridge, Cambridge University Press*, (1985).

[8] SANTOS, P., Tradução Automática, Engenharia da Linguagem. Org. Maria Helena M. Mateus e António Horta Branco. Lisboa, Edições Colibri, (1995).

[9] BERNES-LEE, T. and CAILLIAU, R. *WorldWideWeb: Proposal for a HyperText Project*, (1990).

[10] PAPAZOGLOU, M., GEORGAKOPOULOS, D., *Service-oriented computing, Communications of the ACM: Service-Oriented Computing*, (2003).

[11] FANTINATO, M., TOLEDO, M., GIMENES, I., Arquitetura de Sistemas de Gerenciamento de Processos de Negócio Baseado em Serviços, Rel.T., Unicamp, (2005).

[12] CERAMI, E., *Web Services Essentials: Distributed Applications with XMLRPC, SOAP, UDDI and WSDL*. O'Reilly, (2002).

[13] YE, X., SHEN, Y., *A Middleware for Replicated Web Services. In IEEE International Conference on Web Services (ICWS'05)*, pp 631–638, Orlando, Florida, USA. IEEE Computer Society, (2005).

[14] AYALA, D., BROWNE, C., CHOPRA, V., SARANG, P., APSHANKAR, K., and MCALLISTER,T., *Professional Open Source Web Services*. Wrox Press Ltd., (2002).

[15] NEWCOMER, E., *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison-Wesley Professional, (2002).

[16] ERL, T., *Service-Oriented Architecture: Concepts, Technologies and Design.* Prentice Hall, (2005).

[17] SIMÕES, A., Extração de Recursos de Tradução com Base em Dicionários Probabilísticos de Tradução, Departamento de Informática, Escola de Engenharia, Universidade do Minho, (2008).

[18] IANNACCI, F., *The Linux Managing Model. Department of Information Systems, London School of Economics*, (2004).

[19] http://pt.wikipedia.org/wiki/CVS, acessado em 25/04/2009.

[20] REIS, C., Caracterização de um Processo de Software para Projetos de Software Livre, Universidade de São Paulo, (2003).

[21] MOCKUS, A,. FIELDING, R., HERBSLEB, J., *Two Case Studies of Open Source Software Develpment: Apache and Mozilla*, In: ACM Transactions on Software Engineering and Methodology, (2002).

[22] BENNETH, K. H., et al, *An Architectural Model for Service-Based FlexibeSoftware em Proceedings* COMPSAC 2001 (*Computer Software and Applications*), *IEEE Computer Society Press*, (2001).

[23] MACHADO, J. C., Um estudo sobre o desenvolvimento orientado a serviços. Dissertação de Mestrado, PUC RJ, (2004).

[24] CHARRON-BOST, B., MATTERN, F., TEL, G., *Synchronous, asynchronous and causally ordered communication in Distributed Computing*, Vol. 9, (1996).

[25] Free Software Foundation (FSF), *Categories of Free and Non-Free Software*. Disponível em http://www.gnu.org/philosophy/categories.html. Acessado em 11/05/2009.

[26] SCACCHI, W., *Software Development Practices in Open Software Development Communities: A Comparative Case Study, Institute for Software Research, University of California, Irvine*, (2001).

[27] NUNES, C., STAA, A., Processos de Software Open Source, Monografias em Ciência da Computação, Pontifícia Universidade Católica, Rio de Janeiro, (2007).

[28] REIS, C., Uma Visão Geral do Bugzilla, uma Ferramenta de Acompanhamento de Alterações, XVI Simpósio Brasileiro de Engenharia de Software, (2002).

[29] BRITO, R., FERREIRA, P., SILVA, K., BURERGIO, V., LEITE, I., Uma experiência na implantação de processo em uma fábrica de software livre., Centro de Informática – Universidade Federal de Pernambuco, (2004).

[30] http://softwarelivre.sapo.pt/geral/wiki/RegrasDeLicenciamentoParaGPL, acessado em 11/05/2009.

[31] RAYMOND, E., *The Cathedral and the Bazaar*. (2008). Disponível em: http://catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/, acessado em 20/05/2009.

[32] ALECRIM, E., O Kernel do Linux, (2007). Disponível em: http://www.infowester.com/linuxkernel.php, acessado em 20/05/2009.

[33] http://www.gnu.org/home.pt-br.html, acessado em 26/05/2009.

[34] SOUZA, J., WSFTA: Uma Arquitetura para Tolerância a Faltas em ServiçosWeb, Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Santa Catarina, (2008).

[35] BECKER, A., Claro, D., Sobral, J., *Web Services* e XML Um Novo Paradigma da Computação Distribuída, Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, (2008).

[36] SONDA, G., MONTEZ, C., Uma Proposta de Implementação de Diferenciação de Serviços na Arquitetura de *Web Services*, Departamento de Automação e Sistemas, Universidade Federal de Santa Catarina, (2008).

[37] SPECIA, L., RINO, L., Introdução aos Métodos e Paradigmas de Tradução Automática, Série de Relatórios do Núcleo Interinstitucional de Lingüística Computacional, NILC - ICMC-USP, (2002).

[38] ENGELFRIET, A., *Choosing a Software License*, (2003). Disponível em: http://www.iusmentis.com/computerprograms/licenses/pcactive0203, acessado em 15/11/2009.

[39] FRONDADA, G., BARBOSA, V., GALVAO, R., Arquitetura Orientada a Serviços para Gestão de Processos Acadêmicos na Web, Departamento de Ciência e Tecnologia, Instituto Militar de Engenharia, (2009).

[40] PRESSMAN, R., *Software Engineering: A Practioner's Approach*. McGraw-Hill, 5ª edição, (2001).

[41] BREVE, F., Engenharia para Web, Departamento de Computação, Universidade Federal de São Carlos, (2002).