Radson Antônio Souza Santos

PyTM - Python Task Manager.
Desenvolvimento de um *software* para automação de tarefas utilizando Python.

Monografia de Pós-Graduação *Lato Sensu* apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em Administração em Redes Linux

Orientador
Prof. Ms. Herlon Ayres Camargo

Lavras
Minas Gerais - Brasil
2011

Radson Antônio Souza Santos

PyTM - Python Task Manager. Desenvolvimento de um *software* para automação de tarefas utilizando Python.

Monografia de Pós-Graduação *Lato Sensu* apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em Administração em Redes Linux

Aprovada em *24 de setembro de 2011*

Prof. Dr. Joaquim Quinteiro Uchôa

Prof. Ms. Denilson Vedoveto Martins

Prof. Ms. Herlon Ayres Camargo
(Orientador)

Lavras
Minas Gerais - Brasil
2011

Agradecimentos

Antes de tudo, é necessário agradecer a Deus por ter concedido a graça de enfrentar mais um desafio na caminhada da vida, e com Ele ter chegado à conclusão.

Àqueles que proporcionaram um enriquecimento intelectual durante o curso, os professores e tutores do curso ARL, sinceros agradecimentos. Em especial, ao professor Herlon Camargo, pelos ensinamentos na disciplina Automação de Tarefas que me nortearam na criação deste projeto e que me ajudaram a exercer com profissionalismo o papel de administrador de sistemas GNU/Linux.

Aos colegas do curso ARL, companheiros com quem tive a satisfação de conviver, mesmo que, na maior parte, virtualmente, que, mesmo nos momentos árduos de trabalho, não deixaram de ser irreverentes e amigos. Agradeço também ao colega de trabalho Eric Dias por acreditar e incentivar este projeto.

Por fim, àqueles que foram minha fonte de motivação, que estiveram lado a lado nos momentos de dificuldade e tiveram a paciência de me apoiar. A minha família, Raimundo Antônio, Saralene Bárbara, Taciana Cristina, Bárbara Suellen e Maria Dias. Um agradecimento especial à minha noiva, que vivenciou comigo as emoções durante o curso, Thayanne Moreira.

Com todos vocês, confraternizo a conclusão deste projeto.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

As organizações estão cada vez mais se deparando com novos desafios para manter seus sistemas de TI em pleno funcionamento. O objetivo é gerenciar os componentes de TI sem ter que aumentar os custos. Os gerentes de TI fazem uso de diversas ferramentas e metodologias de automação, porém nem sempre é possível encontrar uma ferramenta totalmente completa. O objetivo deste trabalho é oferecer aos gerentes e administradores uma ferramenta que apresenta uma funcionalidade que quase sempre não está presente nas ferramentas atuais, que é a submissão de tarefas por meio de *scripts*. O *software* desenvolvido chamado de Python Task Manager (PyTM) foi desenvolvido utilizando a linguagem de programação Python. O PyTM faz uso dos conceitos de rede de computadores e implementa a arquitetura cliente-servidor além de trazer o *framework* para desenvolvimento *Web* chamado Django, que também é desenvolvido em Python. Para viabilizar este projeto de *software*, foi utilizada a metodologia de desenvolvimento conhecida como Extreme Programming (XP) em conjunto com a Análise e Desenvolvimento Orientado a Objetos com o apoio dos conceitos da Unified Modeling Language (UML). Utilizando os recursos do Python e com o apoio do sistema gerenciador de banco de dados MySQL, o PyTM apresenta uma solução simples e objetiva para gerenciamento remoto de sistemas através de tarefas expressas por meio de *scripts*.

Palavras-Chave: Automação de tarefas, Gerenciamento de Sistemas, Python, Redes de Computadores.

Capítulo 1

Introdução

1.1 Motivação

A complexidade de TI está em ascensão em muitas organizações. Empresas enfrentam mais escolhas tecnológicas do que nunca, e elas estão contando com um número crescente de sistemas e aplicações de apoio ao negócio.

O aumento na complexidade de TI, em geral, significa mais manutenção de sistemas. Isso, por sua vez, leva a custos mais elevados. Além disso, em um ambiente complexo, as organizações, muitas vezes, não usam seus sistemas de TI de maneira mais eficiente possível e também torna difícil para as organizações fornecerem um ambiente previsível e estável para os usuários.

As organizações podem adotar padrões e *frameworks* baseados nas melhores práticas para ajudar a gerenciar a complexidade de TI. Um *framework* conhecido é a ITIL[1] (IT Infrastructure Library). Um dos passos do ITIL é a recomendação de implementação de uma base de dados de gerenciamento de configuração (CMDB – Configuration Management Database), que inclui a especificação e a identificação de todos os componentes de TI utilizados na organização. Outro *framework* é o COBIT[2] (Control Objectives for Information and Related Technology), um modelo de governança de TI e conjunto de ferramentas de suporte projetado para ajudar organizações a ter melhor controle de TI e segurança no ambiente de informação.

As organizações precisam automatizar seus processos a fim de ganhar eficiência e reduzir a complexidade. Isto pode ser feito automatizando mais tarefas, gerenciando melhor o uso de sistemas de TI, aplicações e redes dentro da orga-

[1] Disponível em: http://www.itil-officialsite.com.
[2] Disponível em: http://www.isaca.org/cobit.

nização. A automação permite que o pessoal de TI possa gastar mais tempo em iniciativas estratégicas e menos tempo com processos manuais.

Este trabalho propõe o desenvolvimento de um *software* que atenda à necessidade de automatização dos processos em um ambiente de TI. O foco principal deste *software* é o gerenciamento de computadores através de tarefas expressas por meio de arquivos de texto com um conjunto de instruções em uma linguagem de programação determinada, também conhecidos como *scripts*. Este *software* deverá permitir ao administrador definir as tarefas a serem executadas e submetê-las a um conjunto de computadores para sua execução.

1.2 Objetivo

Em uma rede de computadores composta por arquiteturas e sistemas operacionais distintos, a administração de sistemas pode se tornar difícil devido à diversidade de situações a serem consideradas.

Centralizar a gerência de computadores em uma rede heterogênea torna-se uma necessidade. Os responsáveis por essas redes precisam de um sistema que possibilite gerenciar todas as estações de forma mais eficiente, rápida e flexível, tendo uma visão da rede por completo. Além disso, esse sistema deve apresentar uma *interface* simples, versátil e, de preferência, remota, favorecendo a mobilidade do administrador.

O objetivo geral deste projeto é o desenvolvimento de um *software* que auxilie a gerência de computadores em uma rede heterogênea. Este *software* deverá atender aos seguintes requisitos iniciais:

- Ser desenvolvido na linguagem de programação Python[3];
- Ser portável, uma vez que as estações de trabalho poderão ser constituídas de arquiteturas e sistemas operacionais distintos;
- Permitir que a gerência dos computadores possa ser feita de forma remota;
- Permitir a submissão de tarefas automatizadas definidas pelo administrador aos computadores na rede;
- Coletar informações sobre os computadores na rede;
- Ter uma interface *Web* de gerenciamento do sistema com mecanismo de autenticação de usuário.

[3]Disponível em: `http://www.python.org`.

Ao final deste projeto, espera-se um protótipo que forneça ao administrador as funcionalidades previstas nos requisitos. Contudo, o *software* não deverá ser utilizado ainda para ambientes reais pois, como primeira versão, será necessário realizar testes e validações que assegurem o funcionamento adequado do mesmo.

1.3 Metodologia

Para atender ao objetivo descrito, este projeto propõe a construção de um sistema distribuído baseado no modelo de comunicação cliente-servidor desenvolvido na linguagem de programação Python. Neste modelo cliente-servidor, o módulo cliente deverá ser responsável por coletar informações do computador hospedeiro e executar as tarefas (*scripts*), e o módulo servidor deverá receber as informações do cliente e registrar numa base de dados, bem como consultar esta base para atribuir as tarefas aos clientes. Por fim, deve-se projetar e desenvolver uma interface *Web* de gerenciamento utilizando o *framework web* Django[4].

A estrutura dos módulos desenvolvidos será concebida após uma análise orientada a objetos e da modelagem desses utilizando alguns diagramas da UML[5] (Unified Modeling Language). Para gerenciar os passos a serem realizados no processo de desenvolvimento, será utilizada a metodologia ágil de programação conhecida como Extreme Programming[6].

A codificação do *software* será feita utilizando um ambiente de desenvolvimento livre, composto por ferramentas para desenvolvimento de *software* disponíveis na maioria das distribuições GNU/Linux. Pretende-se criar um ambiende de desenvolvimento completo para aplicações desenvolvidas em Python.

1.4 Estado da arte

No mercado existem diversas soluções para auxiliar a gerência de estações de trabalho. Geralmente, estas soluções são *software* proprietário, ou seja, requer a aquisição (compra) de uma licença de uso. Porém, alguns não são destinados a redes heterogêneas (nas quais arquiteturas e sistemas operacionais são diferentes) e poucos fornecem funcionalidade de gerenciamento de tarefas expressas por meio de *scripts*, mesmo assim o fazem de forma restrita.

[4]Disponível em: `http://www.djangoproject.com`.
[5]Disponível em: `http://www.uml.org`.
[6]Disponível em: `http://www.extremeprogramming.org`.

Dentre as soluções que atendem ao propósito de gerenciar estações de trabalho, pode-se citar o ZENworks, da Novell[7]; o Desktop Authority, da Script Logic[8]; Synexsys Inventory, da Synexsys[9]; e o Positivo Network Manager, da Positivo[10]. Dentre os citados, são funcionalidades comuns:

- Gerenciamento remoto de usuários, programas, *drivers* de dispositivos;
- Inventário de *hardware* e *software*;
- Monitoramento de sensores dos dispositivos;
- Gerenciamento remoto de processos e serviços.

Entre estas soluções destaca-se o ZENworks, da Novell, que possui uma funcionalidade chamada ZENworks Script Bundle. Esta funcionalidade permite ao administrador escrever *bash scripts* personalizados para Linux, estes *scripts* irão ser executados no *preboot* dos computadores com Linux. Esta funcionalidade provê ao administrador um melhor controle sobre as operações de criação de imagens de dispositivos (ZENworks Imaging) assim como a execução de qualquer tarefa que possa ser expressa em *bash script* (NOVELL, 2010).

No cenário nacional, existem também um conjunto de *software* livre para gerenciamento disponível no portal do Software Público Brasileiro[11] (SPB).

O SPB é utilizado como um dos alicerces para definir a política de uso e desenvolvimento de *software* pelo setor público no Brasil. Tal política compreende a relação entre os entes públicos, em todas as unidades da federação e demais esferas de poder, e destes com as empresas e a sociedade (Portal do Software Público Brasileiro, 2010).

No portal, estão disponíveis diversas soluções para gerenciamento, dentre elas encontra-se o SAGUI[12], desenvolvido pela Serpro, CACIC[13], desenvolvido pela Dataprev (Empresa de Tecnologia e Informações da Previdência Social) e o *software* Koruja[14], desenvolvido pelo Banco do Brasil. Todas as soluções possuem a característica de coletar informações dos equipamentos que compõem um parque computacional (sistema de inventário).

[7]Disponível em: `http://www.novell.com/products/zenworks`.
[8]Disponível em: `http://www.scriptlogic.com`.
[9]Disponível em: `http://www.synexsys.com`.
[10]Disponível em: `http://www.positivoinformatica.com.br`.
[11]Disponível em: `http://www.softwarepublico.gov.br`.
[12]Disponível em: `http://www.softwarepublico.gov.br/dotlrn/clubs/sagui`.
[13]Disponível em: `http://www.softwarepublico.gov.br/dotlrn/clubs/cacic`.
[14]Disponível em: `http://www.softwarepublico.gov.br/dotlrn/clubs/koruja`.

Das soluções disponíveis no SPB, destaca-se o SAGUI (Sistema de Apoio à Gerência Unificada de Informações) por disponibilizar ao administrador a funcionalidade de agendamento de tarefas através de *Bash Script*.

O SAGUI é uma ferramenta de gestão de ativos para ambientes que utilizam sistema operacional GNU/Linux. Foi criado pelo Serpro (Serviço Federal de Processamento de Dados) para automatizar as atividades de instalação, customização, atualização, correção e monitoramento de estações de trabalho e servidores. Com ele, é possível executar *scripts* de correção, customização ou coleta de informações de forma centralizada (Portal do Software Público Brasileiro, 2010).

As soluções de *software* pesquisadas oferecem ao administrador um conjunto de funcionalidades muito relevante, estas auxiliam o administrador na tarefa de manter operacional todas as estações de trabalho de um parque computacional. Evidênciam-se SAGUI e ZENworks por apresentarem funcionalidades similares à proposta neste projeto. Porém, o objetivo deste projeto é oferecer ao administrador maior flexibilidade, permitindo que as tarefas possam ser expressas em qualquer linguagem de *script* (contanto que a estação de trabalho tenha o interpretador necessário) e que possam ser executadas em sistemas operacionais distintos.

1.5 Estrutura do trabalho

Este trabalho está dividido em cinco capítulos. O Capítulo 1 provê uma introdução ao tema proposto, contextualizando o problema em que se insere a solução proposta. Além disso, encontra-se o objetivo deste trabalho bem como a metodologia a ser utilizada e o cenário atual em relação a outros trabalhos com a mesma característica.

O Capítulo 2 apresenta uma revisão de literatura com as principais tecnologias a serem utilizadas no desenvolvimento deste trabalho. Este capítulo traz algumas referências teóricas sobre a linguagem de programação Python bem como do *framework* Django. Serão abordados conceitos de redes de computadores e o desenvolvimento de mecanismos de comunicação em uma rede de computadores por meio da arquitetura cliente-servidor utilizando *sockets*. Os fundamentos apresentados são essenciais para auxiliar no processo de desenvolvimento das aplicações que permitirão uma troca de informações em uma rede de computadores de forma adequada. O capítulo também aborda a utilização dos conceitos de redes na programação utilizando o Python e suas bibliotecas.

Ainda no Capítulo 2, haverá uma explicação sobre uma metodologia de desenvolvimento de *software* intimamente ligada a linguagens de programação dinâmicas como Python. Neste capítulo, é realizada uma breve introdução sobre o

processo de desenvolvimento de *software* e é feito um aprofundamento na metodologia de desenvolvimento Ágil, conhecida como Extreme Programming.

O Capítulo 3 é onde se encontram todas as informações sobre o desenvolvimento do *software* que se chamará Python Task Manager, um gerenciador de tarefas através de *scripts*. O capítulo é uma visão da concepção do *software* em entidades conhecidas como objetos bem como seus relacionamentos e comportamentos, tudo isso irá delinear o processo de codificação do *software*. O capítulo aborda ainda os meios utilizados para concretizar o projeto, como ferramentas, ambiente de execução do projeto, etc.

O Capítulo 4 traz os resultados do desenvolvimento deste projeto, mostrando imagens do produto finalizado. Este capítulo traz também a explanação sobre os benefícios do uso do PyTM.

Por fim, no Capítulo 5, encontram-se as conclusões, planos futuros e contribuições que este trabalho proporcionou.

Capítulo 2

Revisão de literatura

2.1 Considerações iniciais

Neste capítulo, pretende-se apresentar alguns conceitos relacionados às tecnologias e métodos utilizados para a execução deste projeto.

Inicialmente, será abordada a linguagem de programação Python, apresentando, brevemente, a sua trajetória e focando de forma mais abrangente em suas características. Será apresentado também um produto da linguagem Python, que é um dos muitos *frameworks* para desenvolvimento *Web*, conhecido como Django.

Em seguida, serão apresentados conceitos da arquitetura cliente-servidor e o uso de *sockets* na comunicação de computadores, adicionalmente, será feita uma introdução à programação para redes de computadores utilizando o Python. Com estas definições, será possível entender a forma em que o sistema proposto irá se comunicar numa rede de computadores para executar o seu objetivo.

Serão ainda apresentadas algumas definições sobre o processo de desenvolvimento de *software*, assim como os conceitos que sustentam a metodologia de desenvolvimento Extreme Programming, mais conhecida como XP. O XP é uma metodologia ágil para projetos com equipes pequenas e que favorece o desenvolvimento com tecnologias como o Python, que combina muito bem com a metodologia.

Por fim, serão explicadas algumas definições da Análise e Programação Orientado a Objetos que irão embasar e nortear o desenvolvimento deste projeto.

2.2 A linguagem de programação Python

Os computadores podem ser utilizados em diversas áreas de atuação. Por conta disso, existem várias linguagens de programação, cada qual com uma meta dife-

rente. Dentre as metas de cada linguagem, pode-se destacar: aplicações científicas, aplicações comerciais, inteligência artificial, programação de sistemas e *software* para *Web*.

O uso da *Web* explodiu no meio da década de 90 depois do primeiro navegador gráfico ter aparecido. A necessidade de computação associada com documentos HTML, que, por si só, são completamente estáticos, rapidamente, tornou-se crítica. A computação do lado do servidor foi possível com o Common Gateway Interface (CGI), que permitiu a documentos HTML requisitar a execução de programas no lado do servidor, com o resultado da computação retornando para o navegador em forma de documentos HTML. Computação do lado do navegador tornou-se disponível com o advento dos *applets* Java. Ambas destas abordagens estavam sendo lentamente substituídas por novas tecnologias, em grande parte, por linguagens de *script* (SEBESTA, 2006).

As linguagens dinâmicas eram vistas, no passado, apenas como linguagens *script*, usadas para automatizar pequenas tarefas, porém, com o passar do tempo, elas cresceram, amadureceram e conquistaram seu espaço no mercado, a ponto de chamar a atenção dos grandes fornecedores de tecnologia (BORGES, 2010).

Definem-se linguagens como de *scripting* aquelas que são usadas se colocando uma lista de comandos, chamados de *script*, em um arquivo para serem executados. Existem diversas linguagens desta categoria, dentre elas, a linguagem Python.

O Python foi criado no final do ano de 1989 por Guido van Rossum, então, na CWI (Centrum voor Wiskunde en Informatica, o Instituto Nacional de Pesquisa de Matemática e Ciências da Computação), na Holanda. Ela acabou sendo liberada para distribuição ao público no início de 1991 (CHUN, 2006).

Guido van Rossum foi um pesquisador com considerável experiência no projeto da linguagem interpretada ABC, também desenvolvida no CWI, mas ele estava insatisfeito com sua capacidade para ser desenvolvido em algo mais. Tendo usado e parcialmente desenvolvido uma linguagem de alto nível como ABC, voltar para a linguagem C não era uma possibilidade atrativa. Algumas das ferramentas que ele imaginava eram para realizar tarefas gerais na administração de sistemas, e ele também procurava acessar o poder das chamadas de sistema que estavam disponíveis por meio do sistema operacional distribuído Amoeba. Embora van Rossum tenha fornecido uma linguagem específica para o Amoeba, uma linguagem generalizada fazia mais sentido, e, no final de 1989, as sementes do Python foram lançadas (CHUN, 2006).

Python é uma linguagem de programação interpretada, interativa e orientada a objetos. Ela incorpora módulos, exceções, tipagem dinâmica, tipos de dados e classe de alto nível. Python combina notável poder com sintaxe muito clara. Ela tem interface com muitas chamadas de sistema e bibliotecas, assim como para vá-

rios sistemas de janela, e é extensível em C ou C++. O Python também é usado como linguagem de extensão para aplicações que necessitam uma interface programável (Python Software Foundation, 2010).

A implementação padrão do Python é escrita em ANSI C portável, e ele compila e funciona em praticamente todas as principais plataformas atualmente em uso. Por exemplo, os programas em Python hoje rodam em tudo, desde PDAs a supercomputadores. Como uma lista parcial, Python está disponível em (LUTZ, 2009):

- Sistemas Linux e Unix;
- Microsoft Windows e DOS (todas as versões modernas);
- Mac OS (ambos OS X e Classic);
- BeOS, OS/2, VMS, e QNX;
- Sistema de tempo real como VxWorks;
- Supercomputadores Cray e mainframes IBM;
- PDAs rodando Palm OS, PocketPC, e Linux;
- Telefones celulares rodando Symbian OS e Windows Mobile;
- Consoles de jogos e iPods.

Como o interpretador da linguagem, os módulos da biblioteca padrão que vêm com Python são implementados para ser tão portáteis, através dos limites da plataforma, quanto possível. Os programas em Python são automaticamente compilados para o formato portável chamado de *byte code*, essa característica faz com que programas Python usando a linguagem básica e a biblioteca padrão executem o mesmo no Linux, Windows e outros sistemas operacionais com um interpretador Python.

Python é uma linguagem de programação extremamente poderosa e dinâmica, que é usada em uma variedade de domínios de aplicação. Python é, frequentemente, comparada ao Tcl, Perl, Ruby, Scheme ou Java. Algumas das suas principais características distintivas incluem (Python Software Foundation, 2010):

- Sintaxe muito clara e legível;
- Forte capacidade de introspecção;
- Intuitiva orientação a objeto;

- Expressão natural de código procedural;
- Modularidade completa, suportando hierárquica de pacotes;
- Exceção baseada em manipulação de erro;
- Muito elevado nível de tipos de dados dinâmico (VHLL - Very High Level Language);
- Extensa biblioteca padrão e módulos de terceiros para, praticamente, qualquer tarefa;
- Extensões e módulos facilmente escritos em C, C++ (ou Java para Jython ou .NET para IronPython);
- Embutido dentro de aplicações como uma interface de *script*.

O modelo de classes do Python suporta noções avançadas, tais como polimorfismo, sobrecarga de operadores e herança múltipla. Além de servir como um poderoso dispositivo de estruturação e reutilização de código, a natureza de orientação a objeto do Python o torna ideal como uma ferramenta de *scripting* para linguagens de sistemas orientadas a objeto como C++ e Java. O Python suporta tanto o paradigma de programação procedural como o orientado a objeto.

Um dos atributos do Python é a sua biblioteca padrão. Geralmente, é utilizado o termo "baterias incluídas"em referência ao Python, isto significa que a biblioteca padrão permite ao desenvolvedor realizar toda sorte de tarefas sem ter que buscar em outro lugar módulos para ajudar. Python inclui funcionalidades para expressões regulares; *sockets*; *threads*; funcionalidade para data/tempo; analisadores XML; analisadores para arquivos de configuração; funcionalidades para manipulação de arquivos e diretórios; persistência de dados; capacidade para unidades de testes; bibliotecas clientes para os protocolos HTTP, FTP, IMAP, SMTP e NNTP. Portanto, uma vez que o Python está instalado, módulos para suporte de todas essas funções serão importados pelos *scripts* do desenvolvedor (GIFT; JONES, 2008).

De acordo com (PILGRIM, 2004), tudo na linguagem Python é objeto. Ele cita que a introspecção é como se o código pensasse nos outros módulos e funções na memória como objetos, pegando informações sobre eles e manipulando eles.

A sintaxe do Python não é baseada diretamente em qualquer linguagem usada comumente. Ela é fortemente tipada, mas dinamicamente tipada. Em vez de vetores, Python inclui três tipos de estrutura de dados: listas, listas imutáveis, que podem ser chamadas tuplas e hashes, que são chamados de dicionários (SEBESTA, 2006).

Python é distribuída sob uma licença própria (compatível com a GPL), que impõe poucas restrições. É permitida a distribuição, comercial ou não, tanto da linguagem quanto de aplicações desenvolvidas nela, em formato binário ou código fonte, bastando cumprir a exigência de manter o aviso de Copyright da PSF (Python Software Foundation) (PYTHONBRASIL, 2010).

Algumas distribuições GNU/Linux utilizam o Python em suas rotinas do sistema operacional e também em ferramentas de gerenciamento. Pode-se citar o Gentoo Linux, que utiliza o mecanismo de gerência de pacotes chamado Portage[1], bem como o sistema de gerenciamento de pacotes YUM[2] (Yellowdog Updater Modified) da distribuição Red Hat Enterprise Linux e seus derivados. A Red Hat utiliza o Python também no seu conhecido programa de instalação do sistema operacional chamado Anaconda[3].

2.3 Django, *framework* para desenvolvimento *Web* em Python

Django é o nome do *framework* para desenvolvimento de aplicações *Web* em Python de alto nível que encoraja um desenvolvimento rápido, limpo e com *design* pragmático. Foi criado para o desenvolvimento do *website* do jornal Lawrence Journal-World[4] por Adrian Holovaty, Simon Willison e Jacob Kaplan-Moss.

Como foi desenvolvido e usado para as operações de um jornal *Web*, o Django é bem apropriado para desenvolvimento de sistemas gerenciadores de conteúdos. Foi projetado a partir da escrita rápida para lidar com prazos intensivos de uma sala de redação e exigências rigorosas de experientes desenvolvedores *Web*. Essa situação fez o Django focalizar na automatização aderindo ao princípio DRY (Don't Repeat Yourself), esse princípio refere-se a evitar a duplicação de código (Django Project, 2010).

Como convém a sua herança da redação de um jornal, Django anuncia-se como "o *framework Web* para perfeccionistas com prazos". Seu núcleo é um conjunto de sólidas e bem testadas bibliotecas abrangendo todos os aspectos repetitivos do desenvolvimento *Web* (BENNETT, 2009):

- Um mapeador objeto-relacional, que é uma biblioteca que sabe como o banco de dados se parece, como o código se parece, e como fazer a ponte entre eles sem repetir códigos SQL escritos a mão;

[1] Disponível em: `http://www.gentoo.org/doc/en/handbook`.
[2] Disponível em: `http://yum.baseurl.org`.
[3] Disponível em: `http://fedoraproject.org/wiki/Anaconda`.
[4] Disponível em: `http://www.ljworld.com`.

- Um conjunto de bibliotecas HTTP que sabe como analisar os pedidos *Web* recebidos; como entregá-las em um padrão, formato fácil de usar, e como transformar os resultados dos códigos em respostas bem formadas;
- Uma biblioteca de roteamento de URL que permite definir exatamente as URLs que devem ser mapeadas para as partes relevantes do código;
- Uma biblioteca de validação que ajuda o desenvolvedor a mostrar formulários nas páginas *Web* e a processar os dados submetidos pelos usuários;
- Um sistema de *templates* que permite, mesmo a não programadores, escreverem HTML misturado com dados gerados pelo código e apenas a quantidade certa da lógica de apresentação.

2.3.1 Funcionamento do Django

Quando um endereço da *Web* é acessado no navegador, é realizada uma ação conhecida como requisição (`HttpRequest`). Uma requisição possui diversas informações, sendo a URL (Uniform Resource Locator) uma das mais importantes. A URL é composta, respectivamente, pelo protocolo, o domínio, a porta, o caminho e os parâmetros.

```
http://localhost:8000/admin/?modelo=Camaro&ano=2010
```

Figura 2.1: Exemplo de URL.

A Figura 2.1 ilustra um exemplo de URL, sendo que seus componentes são descritos a seguir:

- Protocolo: `http`
- Domínio: `localhost`
- Porta: `8000`
- Caminho: `/admin/`
- Parâmetros: `modelo = Camaro e ano = 2010`

Ao chegar no Django, uma requisição passa primeiro no *Handler* para, então, ser repassada para os *middlewares*. Quando a resposta está retornando do Django, o *Handler* é o último passo.

Os *Middlewares* são pequenos trechos de código que analisam a requisição na entrada e a resposta (`HttpResponse`) na saída. Após passar pelos *middlewares*, a requisição irá conter outras informações além das originais, como usuário autenticado, sessão atual, dentre outros.

Um dos *middlewares* faz com que toda a segurança e autenticação de usuários seja feita. Outro adiciona a função de sessão, uma forma de memorizar o que o usuário está fazendo para atendê-lo melhor. Há, ainda, outro *middleware*, que memoriza as respostas no Cache, pois, se caso a próxima requisição tenha o mesmo caminho e os mesmos parâmetros, ele retorna a resposta memorizada, evitando o trabalho de ser processada novamente pela *view* (BRANDAO, 2009).

Após passar pelos *middlewares*, a requisição chega ao *URL Dispatcher*. Este componente verifica o endereço e verifica o arquivo `urls.py` do projeto a fim de encontrar a *view* que será chamada para responder a requisição. A *View* é uma função escrita pelo programador (armazenadas no arquivo `views.py`) que recebe uma requisição (`HttpRequest`), analisa a requisição e retorna uma resposta (`HttpResponse`) que, geralmente, é uma página.

O Django possui um recurso chamado de *Generic Views*, este recurso é um conjunto de *Views* prontas com funcionamento já conhecido. São úteis para blogs, cadastros, recuperação de senhas, autenticação de usuários, realizar *Logoff* do sistema, redirecionamento, etc.

Um *template* Django é uma sequência de texto que se destina a separar a apresentação de um documento de seus dados. Um *template* define espaços e vários pedaços de lógica básica (*tags*) que regulam o modo como o documento deve ser exibido. Normalmente, os modelos são utilizados para produzir o HTML, mas os modelos Django são igualmente capazes de gerar qualquer formato baseado em texto (HOLOVATY; KAPLAN-MOSS, 2009).

Para persistência e consulta de informações do banco de dados, não é necessário conhecimento da linguagem de consulta do banco, que, geralmente, é o SQL (Structured Query Language). O Django prove o mapeador objeto-relacional (ORM - Object-relational mapping) que interpreta a programação de acesso aos objetos no código em Python e converte em consultas ao banco de dados. O arquivo responsável por este mapeamento é o `models.py`.

Em resumo, o que se tem no Django é uma separação de conceitos:

- O Modelo contém a descrição das tabelas da base de dados, representados através de classes Python;

- A Visão contém a lógica do negócio da página, onde as requisições são recebidas e as respostas são retornadas;

- O Controle são todos os outros mecanismos entre a requisição e a resposta, como: *handler*, *middlewares* e *URL dispatcher*.

A Figura 2.2 resume a estrutura de funcionamento do Django (BRANDAO, 2009).

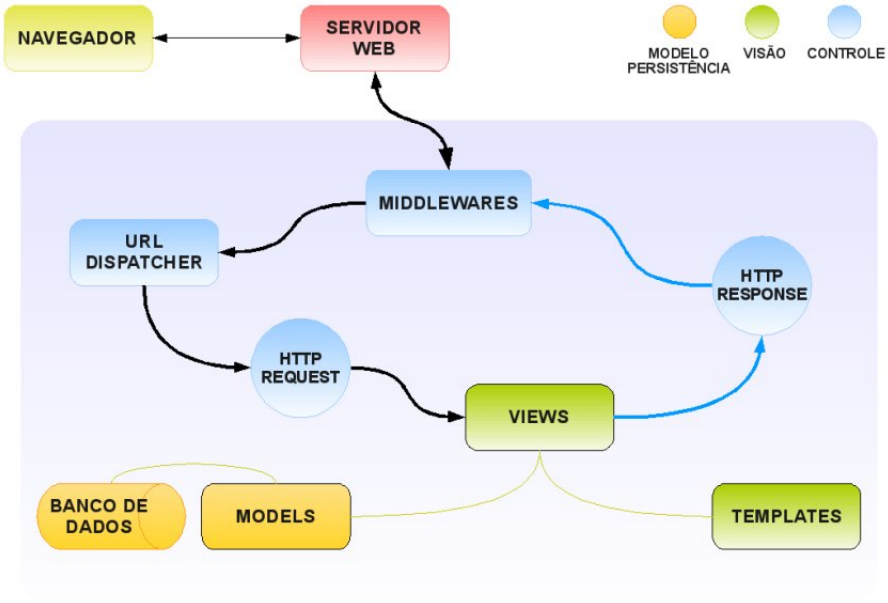

Figura 2.2: Estrutura de funcionamento do Django.

Usadas juntas, essas peças livremente seguem um padrão chamado *Model-View-Controller* (MVC). O MVC é um caminho de desenvolvimento de *software* em que o código para definir e acessar os dados (o modelo) está separado da lógica de percurso da requisição (o controle), que, por sua vez, é separada da interface do usuário (a visão) (HOLOVATY; KAPLAN-MOSS, 2009).

Django é um *framework* MVC. No entanto, o controlador é chamado de "visão", e a visão é chamada de "*template*". A visão de Django é o componente que recupera e manipula dados, enquanto que o *template* é o componente que apresenta dados para o usuário. Por esta razão, o Django é conhecido, as vezes, como um *framework* MTV (onde MTV está para o modelo, *template* e visão). Esta terminologia diferente não muda o fato de que o Django é um *framework* MVC, nem afeta a forma como as aplicações são desenvolvidas (HOURIEH, 2009).

2.4 Desenvolvimento de aplicações para rede

Nos primórdios da Internet, as pessoas estavam interessadas em fazer os computadores se comunicarem entre si. Muitos métodos foram desenvolvidos, uns continuam até hoje e outros, não. Contudo, o mais popular de todos os protocolos é conhecido como Transmission Control Protocol/Internet Protocol (TCP/IP). O TCP/IP é o protocolo padrão que permite computadores por todo o mundo se comunicarem através da Internet (GOERZEN, 2004).

A Internet é baseada na arquitetura cliente-servidor. Existem muitas maneiras de realizar a comunicação entre o cliente e o servidor, porém a mais comum é por meio de uma interface conhecida como *socket*.

2.4.1 Arquitetura cliente-servidor

Na arquitetura cliente-servidor, um computador pode se comunicar com outro para solicitar um serviço. Uma vez que o propósito de uma rede é oferecer serviços para que os computadores se comuniquem, é necessário que estes estejam executando algum tipo de programa que permita essa comunicação. Estes programas aplicativos devem oferecer um serviço para que outro programa possa solicitar o serviço. Entendendo os aplicativos como entidades que se comunicam, elas podem ser definidas como duas entidades distintas, cliente e servidor.

A entidade "cliente" é um programa executado em um computador local que solicita serviços de um computador remoto servidor. O canal de comunicação é estabelecido através do endereço IP e de um número de porta conhecida do computador remoto. Enquanto o canal de comunicação está ativo, o cliente pode solicitar e receber respostas do servidor repetidas vezes até o fim do processo.

Por sua vez, a entidade "servidor" é um programa executado em um computador remoto oferecendo serviços aos clientes. O servidor disponibiliza portas de entrada para receber requisições dos clientes. O servidor deve ser executado de forma ininterrupta para atender aos clientes de forma iterativa, onde cada requisição é tratada por vez ou de forma concorrente, onde diversas solicitações podem ser tratadas simultaneamente.

No programa servidor, os protocolos User Datagram Protocol (UDP) ou Transmission Control Protocol (TCP) podem ser utilizados como protocolos da camada de transporte. O tipo de protocolo utilizado é característica determinante do modo de operação do servidor, este pode operar sem conexão (UDP) ou orientado à conexão (TCP).

As aplicações servidoras que usam o UDP normalmente são iterativas, o servidor processa uma solicitação por vez. Os servidores usam um único número de

porta para este propósito. Todos os pacotes que chegam a esta porta aguardam na linha para serem servidas (FOROUZAN, 2006).

As aplicações servidoras que usam o TCP, normalmente, são concorrentes. Isto significa que o servidor pode servir a muitos usuários ao mesmo tempo. Uma conexão é estabelecida entre o servidor e cada cliente, e a conexão permanece aberta até que todo fluxo de *bytes* seja processado (FOROUZAN, 2006).

Em um servidor concorrente, a aplicação não pode utilizar apenas portas pré-destinadas, pois cada conexão vai requerer um número de porta e podem ser estabelecidas diversas conexões simultaneamente. Para resolver este problema, uma porta é conhecida e as outras portas são temporárias. Dessa forma, o servidor fica escutando em uma porta conhecida e os clientes estabelecem conexão nesta porta, em seguida, o servidor atribui uma porta temporária para a conexão com o cliente e libera a porta para receber novas conexões, como ilustrado na Figura 2.3.

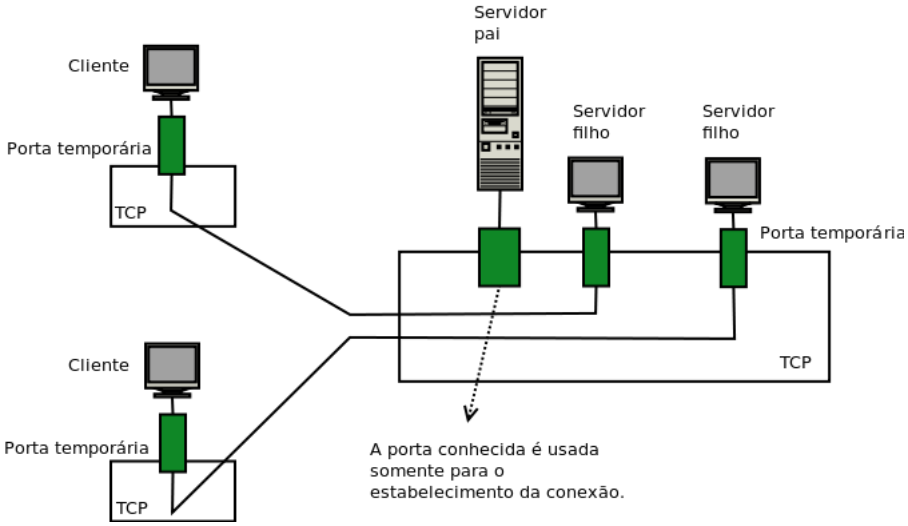

Figura 2.3: Servidor concorrente orientado à conexão.

2.4.2 Comunicação via *sockets*

A interface *socket* foi desenvolvida originalmente no sistema operacional UNIX e define um conjunto de chamadas ao sistema (os procedimentos) que são uma extensão das chamadas usadas no UNIX para acessar arquivos. O *socket* é definido no sistema operacional como uma estrutura, esta, por sua vez, é utilizada na programação de comunicação entre dois processos.

Em um ambiente TCP/IP, a interface *socket* pode ser definida a partir de três tipos:

- *Stream socket*: Utilizado com um protocolo orientado à conexão (TCP);
- *Packet socket*: Utilizado com um protocolo sem conexão (UDP);
- *Raw socket*: Protocolos como o ICMP (Internet Control Message Protocol) ou OSPF (Open Shortest Path First) utilizam diretamente os serviços do protocolo IP.

O conjunto de primitivas de transporte *socket* é amplamente utilizado em programação para a Internet. Essas primitivas são detalhadas na Tabela 2.1.

Tabela 2.1: Primitivas do *Socket*.

Primitiva	Descrição
`SOCKET`	Criar um novo ponto final de comunicação.
`BIND`	Anexar um endereço local a um *socket*.
`LISTEN`	Anunciar a disposição para aceitar conexões; mostra o tamanho da fila.
`ACCEPT`	Bloquear o responsável pela chamada até uma tentativa de conexão ser recebida.
`CONNECT`	Tentar estabelecer uma conexão ativamente.
`SEND`	Enviar alguns dados através da conexão.
`RECEIVE`	Receber alguns dados da conexão.
`CLOSE`	Encerrar a conexão.

As quatro primeiras primitivas na lista são executadas pelos servidores nessa mesma ordem. A primitiva `SOCKET` cria um ponto final e aloca espaço de tabela para ele na entidade de transporte. Os parâmetros de chamada especificam o formato de endereçamento a ser usado, o tipo de serviço desejado, e o protocolo. Uma chamada `SOCKET` bem-sucedida retorna um descritor de arquivo comum que será usado nas chamadas subsequentes, exatamente como uma chamada `OPEN` (TANENBAUM, 2003).

Os *sockets* recém-criados não têm endereço de rede; esses endereços são atribuídos através de uma primitiva `BIND`. Uma vez que um servidor tenha designado um endereço para um *socket*, os clientes remotos já podem se conectar a ele.

Em seguida, a chamada `LISTEN` aloca espaço para a fila de chamadas recebidas, caso vários clientes tentem se conectar ao mesmo. A chamada `LISTEN` não é bloqueante.

Para bloquear a espera por uma conexão de entrada, o servidor executa uma primitiva `ACCEPT`. Quando chega uma unidade de dados do protocolo de transporte (TPDU[5]) solicitando uma conexão, a entidade de transporte cria um novo *socket* com as mesmas propriedades do *socket* original e retorna um descritor de arquivo para ele. Em seguida, o servidor pode desviar um processo ou um *thread* para tratar a conexão no novo *socket* e voltar a esperar pela próxima conexão no *socket* original. `ACCEPT` retorna um descritor de arquivo normal, que pode ser usado para ler e gravar da maneira padrão, como no caso de arquivos (TANENBAUM, 2003).

No cliente, uma primitiva `SOCKET` também é utilizada na criação do *socket*, porém, a primitiva `BIND` não é necessária, pois o endereço usado não é importante para o servidor. A primitiva `CONNECT` bloqueia o responsável pela chamada e inicia o processo de conexão. Quando a conexão é concluída, o processo cliente é desbloqueado e a conexão é estabelecida. Depois disso, ambos os lados podem usar as primitivas `SEND` e `RECV` para transmitir e receber dados através da conexão *full-duplex*.

O encerramento da conexão com *sockets* é simétrico. Quando ambos os lados tiverem executado uma primitiva `CLOSE`, a conexão será encerrada. Todo este fluxo pode ser acompanhado na Figura 2.4.

2.4.3 Programação para rede com Python

A Internet trabalha enviando lotes de tráfego através de linhas compartilhadas. O TCP permite compartilhamento quebrando um fluxo de dados em pequenos pacotes, que são enviados por meio da Internet (possivelmente intercalados com pacotes de outras aplicações), e, então, remontando-os na outra extremidade. Ao tornar os pacotes pequenos, a conexão da Internet é somente amarrada com o envio de cada pedaço de dados em uma pequena quantidade de tempo, e pacotes de outras aplicações podem ser enviados também (GOERZEN, 2004).

Programas para rede com Python, normalmente, encontram-se em duas opções para se programar: utilizando os módulos dos protocolos (como, por exemplo, HTTP e FTP) disponíveis na biblioteca padrão do Python, ou desenvolver programas que requerem a escrita de um protocolo de comunicação a partir do zero.

O Python provê uma interface completa para a interface *socket* subjacente ao sistema operacional através do módulo chamado `socket`. O Python também provê acesso a serviços como SSL (Secure Socket Layer) e TLS (Transport Layer Security) para canais de comunicação encriptados e autenticados.

A biblioteca padrão do Python tem todo o suporte para arquivos e objetos que são tratados como arquivos. O Python provê métodos para manipulação de obje-

[5]TPDU – Transport Protocol Data Unit

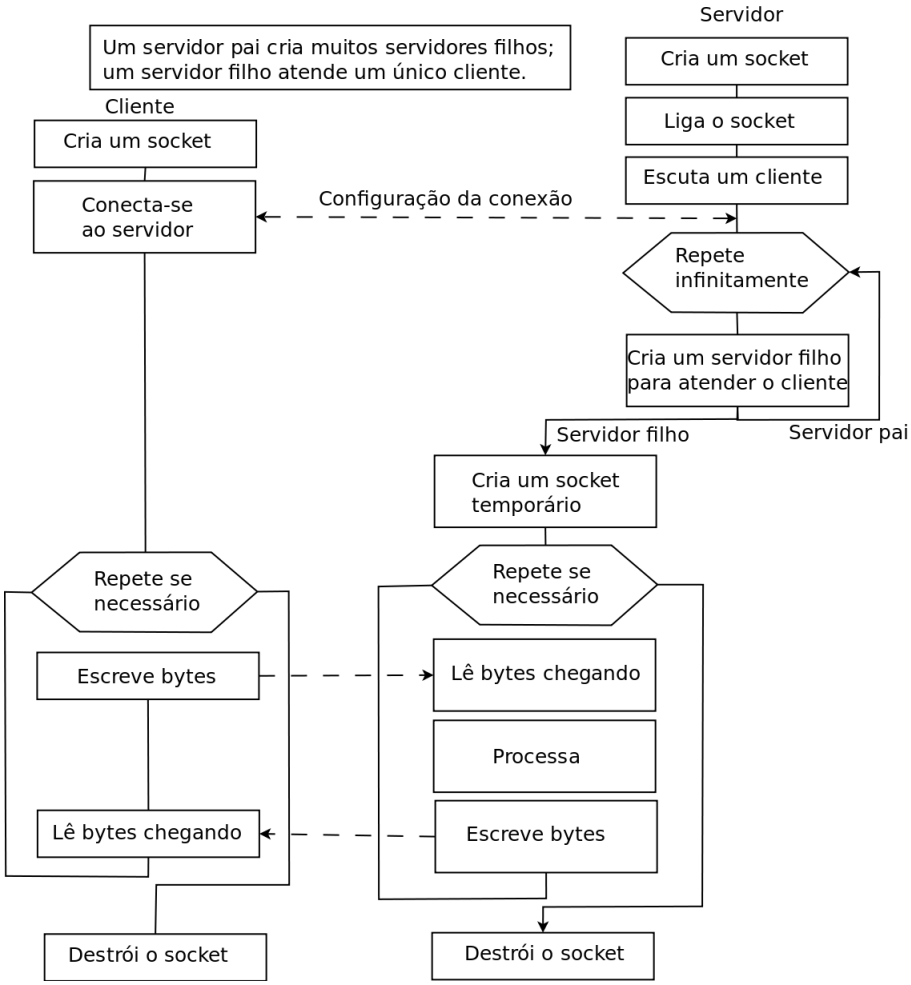

Figura 2.4: Fluxograma *socket* para servidor concorrente orientado à conexão.

tos com característica de arquivos como `readline()`, `write()`, `read()` e outros similares. O objeto Socket não implementa a mesma interface, contudo, o Python dispõe do método `makefile()` para o objeto Socket que gera uma abstração do `socket` como sendo um arquivo.

A Figura 2.5 mostra um exemplo de um programa implementando um servidor *socket* através do módulo `socket` do Python e utilizando a chamada ao método `makefile()` do mesmo.

No exemplo da Figura 2.5, a parte principal do programa começa com a importação do módulo `socket`. Este irá permitir a criação de um *socket* por meio da

```
1  #!/usr/bin/env python
2  # Servidor socket simples - server.py
3
4  import socket
5
6  host = '' # Liga o socket para todas as interfaces
7  porta = 8000
8
9  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10 s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
11 s.bind((host, porta))
12 s.listen(1)
13
14 print "Servidor esta rodando na porta %d; Ctrl-C para terminar." % porta
15
16 while 1:
17     sock, end = s.accept()
18     arquivo = sock.makefile('rw', 0)
19     arquivo.write("Bem vindo, " + str(end) + "\n")
20     arquivo.write("Entre com um texto: ")
21     linha = arquivo.readline().strip()
22     arquivo.write("Foram digitados %d caracteres.\n" % len(linha))
23     arquivo.close()
24     sock.close()
```

Figura 2.5: Exemplo de servidor *socket* com Python.

chamada `socket.socket()`. Este método recebe dois parâmetros, um definindo a família de endereços IPv4 (`AF_INET`) e o outro definindo o tipo de comunicação utilizando um fluxo de dados confiável através do TCP (`SOCK_STREAM`).

Em seguida, já com o objeto `socket` instanciado, algumas opções adicionais são passadas para esse com a chamada do método `setsockopt()`, que define opções para o nível do *socket* (`SOL_SOCKET`), seguido de diretiva, que permite a reutilização do endereço local (`SO_REUSEADDR`).

A chamada ao método `bind()` faz a ligação do *socket* para escutar na porta 8000 em qualquer interface, por isso, a variável `host` foi declarada como *string* vazia.

O próximo método chamado é o `listen()`, que diz ao servidor para ficar aguardando conexões dos clientes, o parâmetro recebido pelo método diz o número máximo de conexões pendentes, o sistema operacional deve colocar na fila antes de recusar as conexões.

Dentro do laço *while*, a chamada `accept()` faz o programa parar naquele ponto até receber a conexão de um cliente. Este método retorna um par `(sock,`

`end)`, onde `sock` é um novo objeto conexão que pode ser usado para enviar e receber dados da conexão e `end` é o endereço do *socket* na outra ponta da conexão (IP e porta).

O método `makefile()` cria uma associação do *socket* com um arquivo permitindo escrita e leitura. Neste, a escrita e leitura de informações segue como em um arquivo através do `write()` e `readline()`. O servidor imprime uma mensagem de boas-vindas, lê informação do cliente e, então, imprime de volta uma resposta.

Ao final, o método `close()` trata de fechar o arquivo e o *socket* para evitar que o servidor acumule conexões inativas de clientes e para que estes saibam quando foi finalizada a comunicação.

Para realizar um teste básico deste servidor de exemplo, basta salvá-lo num arquivo `server.py` e executar como `./server.py` em um terminal. Em outro terminal, para acessar o servidor, utiliza-se o comando `telnet`, como na Figura 2.6.

```
$ telnet 127.0.0.1 8000
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Bem vindo, ('127.0.0.1', 52252)
Entre com um texto: Hello World
Foram digitados 11 caracteres.
Connection closed by foreign host.
```

Figura 2.6: Exemplo de acesso ao servidor via `telnet`.

2.5 Desenvolvimento ágil de *software*

O desenvolvimento ágil é um termo dado a toda uma classe de metodologias de desenvolvimento iterativo de *software*. Sua característica unificadora é um foco em ciclos curtos de desenvolvimento, na escala de semanas e não meses. Cada ciclo de desenvolvimento, referida como uma iteração ou *sprint*, produz um produto de trabalho.

Métodos ágeis enfatizam acomodar mudanças, comunicação em grupo, e iterativo escopo e desenvolvimento. Eles tentam livrar-se do excesso de processos. Algumas delas são simplesmente descartadas, algumas são substituídas por outras práticas. Metodologias ágeis variam de Extreme Programming (XP), que se concentra quase que exclusivamente no desenvolvedor e técnicas de desenvolvimento, para o Dynamic Systems Development Method (DSDM), que se concentra quase

que totalmente sobre os processos, mas todos eles têm similaridades (YOUNKER, 2008).

2.5.1 Processo de desenvolvimento de *software*

O processo de *software* é definido com uma sequência coerente de tarefas ou fases para a construção de um *software*. As fases quem compõem um processo de *software* são (REIS, 2001):

- **Especificação de Requisitos**: tradução da necessidade ou requisito operacional para uma descrição da funcionalidade a ser executada;
- **Projeto de Sistema**: tradução destes requisitos em uma descrição de todos os componentes necessários para codificar o sistema;
- **Programação (Codificação)**: produção do código que controla o sistema e realiza a computação e lógica envolvida;
- **Verificação e Integração (*Checkout*)**: verificação da satisfação dos requisitos iniciais pelo produto produzido;
- **Manutenção e Evolução**: o *software*, em geral, entra em um ciclo iterativo que abrange todas as fases anteriores.

Os principais modelos de processo de *software* são:

- **Cascata**: Cada fase é realizada por completo e os resultados servem como entrada para a próxima fase;
- **Espiral**: As fases são realizadas através de ciclos evolutivos resultando num produto incremental;
- **Ágil**: Desenvolvimento em iterações (curtos períodos de desenvolvimento) que resultam em uma nova versão do *software* a cada iteração.

Para a execução deste projeto, será utilizada a metodologia de desenvolvimento ágil Extreme Programming (XP).

2.5.2 Extreme Programming

O XP é uma metodologia de desenvolvimento ágil para pequenas e médias equipes de desenvolvimento de *software* diante de mudanças rápidas de requisitos. O XP consiste de quatro partes: valores, princípios, atividades e práticas.

Figura 2.7: As partes do XP.

A Figura 2.7 demonstra como estão constituídas essas partes, com atividades de acontecem em torno ou durante todo o ciclo de vida.

É possível dizer que o XP emergiu de uma série de cooperações, tendências e dos melhores sistemas inventados ou criados. De acordo com esta visão orgânica do desenvolvimento do *software*, o XP é fundamentado em valores e guiado por princípios. Neste núcleo, as práticas e as atividades de XP estão sustentadas. A Figura 2.8 demonstra como o XP é uma sinergia de quatro componentes que foram desenvolvidos das influências-chave, e o resultado deste é qualidade para o cliente.

O XP é dirigido por um conjunto de valores compartilhados que estabelecem o tom para o desenvolvimento em XP.

- **Simplicidade**: Significa que, com simplicidade em mente, a equipe de desenvolvimento deve somente implementar o que o cliente necessita realmente não o que espera que pode necessitar;

- **Comunicação**: O foco está na comunicação oral sem documentos, relatórios e planos. Sem uma comunicação constante entre todos os membros da equipe, a colaboração enfraquecerá e morrerá. XP tem as práticas, tais como a programação em pares, que requerem comunicação para trabalhar;

- **Feedback**: O valor do *feedback* é vital para o sucesso de qualquer projeto de *software*. Com XP, as perguntas sobre o estado do sistema são respondidas através de constante e concreto *feedback*;

Figura 2.8: A evolução do XP para criar resultados da qualidade.

- **Coragem**: A coragem é a confiança para trabalhar rapidamente e para reconstruir se requerido. A coragem em XP deve ser pensada no contexto dos outros três valores; sem estes, a coragem conduz ao caos.

Baseados no núcleo e nos valores que são o núcleo do XP, têm-se cinco princípios fundamentais que guiam o desenvolvimento de *software*.

- **Feedback Rápido**: Significa que os desenvolvedores usam ciclos curtos de *feedbacks* para rapidamente saber se seu produto atende as necessidades do cliente na data da reunião;
- **Assumir Simplicidade**: Tratar cada problema como se pudesse ser resolvido simplesmente;
- **Mudança Incremental**: Resolver problemas com uma série de mudanças pequenas. Isto se aplica ao planejamento, ao desenvolvimento e ao projeto;
- **Aceitar mudanças**: Adotar uma estratégia que preserve opções ao resolver problemas urgentes;
- **Trabalho de qualidade**: A qualidade do trabalho nunca pode ser comprometida. XP promove a importância do codificar e de testar.

No XP, as atividades funcionam durante todo o ciclo de vida.

- **Escutar**: O XP é baseado na comunicação e tem as práticas que requerem escutar ativamente. Além de simplesmente de dizer que os colaboradores devem escutar os clientes, XP tem as práticas que dirigem e guiam para uma comunicação melhor;
- **Testar**: É uma etapa integral durante todo o processo. Isto é até ao ponto em que os desenvolvedores escrevem testes antes que desenvolvam o código. O resultado é uma melhor qualidade do código;
- **Codificar**: Com um pequeno projeto escrito, desenvolvedores XP escrevem o código que expressa intencionalmente seu significado. Pela mera leitura do código fonte, outros membros da equipe podem compreender a lógica, os algoritmos e o fluxo;
- **Projetar**: O projeto é não estacionário ou atribuído a um papel, mas é baseado em equipes e dinâmico. Em um sentido, o *software* está sempre em um estado do projeto. Em vez de ignorar este fato, limitando atividades de projeto, XP aceita a evolução natural do sistema.

O XP expressa suas atividades em 12 práticas principais. Estas práticas são o que as equipes de XP usam cada dia desenvolver sistemas.

- **Jogo do Planejamento**: Deduzir rapidamente um plano de alto nível para a liberação ou a iteração seguinte;
- **Liberações pequenas**: Os ciclos de XP consistem nas liberações frequentes do *software*;
- **Metáfora**: É a visão, os termos e a língua comuns usada para descrever o projeto;
- **Projeto simples**: O projeto deve ser simples. Significa que o código faz a coisa a mais simples que poderia fazer;
- **Testes**: Usar testes automatizados;
- ***Refactory***: Significa melhorar o projeto de código existente sem mudar o comportamento do sistema;
- **Programação em pares**: Dois desenvolvedores se sentam na mesma máquina e compartilham tarefas de desenvolvimento;
- **Posse coletiva**: Permite a qualquer um melhorar, ou mudar a qualquer hora, qualquer parte do código;

- **Integração contínua**: Os componentes do sistema são construídos ou integrados, muitas vezes, a cada dia;
- **Semana de trabalho de 40 horas**: O XP exige horas regulares de um trabalho para assegurar a qualidade e desempenho;
- **Cliente no local**: O cliente ou um representante está no local de desenvolvimento e faz parte da equipe;
- **Padrões de codificação**: É um grupo de convenções que todos concordam seguir no desenvolvimento.

A produtividade e expressividade das linguagens dinâmicas se encaixam perfeitamente com as metodologias ágeis, que nasceram do desenvolvimento de *software* de código aberto e defendem um enfoque mais pragmático no processo de criação e manutenção de *software* do que as metodologias mais tradicionais (BORGES, 2010).

2.6 Análise e programação orientada a objetos

Na engenharia, o desenvolvimento de um sistema implica sempre em projetar sua estrutura, provendo uma decomposição adequada em partes separadas e as relações entre elas. Esta abordagem permite a engenheiros resolver a complexidade de um sistema aplicando a abordagem dividir para conquistar – que é recursivamente focando em subsistemas limitados e na interação entre eles (LUCIA *et al.*, 2008).

A orientação a objetos consiste em conceber um sistema computacional como um todo orgânico, formado por objetos que se relacionam entre si. Esse enfoque por ser aplicado tanto à análise de sistemas quanto à programação, e essa é uma das principais vantagens da orientação a objetos: a mesma metodologia serve tanto para a definição lógica do sistema quanto para a sua implementação (CORREIA; TAFNER, 2006).

Na análise e programação orientada a objetos, o processo de análise diz respeito à identificação dos objetos existentes no universo que se pretende automatizar juntamente com seus atributos e ações. Na programação orientada a objetos, utiliza-se o resultado do processo de análise transcrevendo as entidades identificadas em estruturas de dados que simulam o comportamento dos objetos.

A análise orientada a objetos é uma especificação técnica semiformal para o paradigma orientado a objetos. Por ser uma técnica semiformal, uma parte intrínseca de cada técnica de análise orientada a objeto é a notação gráfica associada

com essa técnica. Portanto, aprender a utilizar uma técnica específica significa aprender a notação gráfica pertinente para aquela técnica (SCHACH, 2004). Para solucionar este impasse, uma notação comum foi publicada para qualquer técnica de análise orientada a objetos, esta notação chama-se UML.

A Unified Modeling Language (UML) é uma notação gráfica para expressar projetos orientados a objetos. Ela é chamada de linguagem de modelagem e não uma notação de projeto, pois permite que representar vários aspectos do sistema, não apenas o projeto que tem de ser implementado. Para um projeto, uma especificação das classes que existem no sistema pode ser suficiente. No entanto, enquanto se modela, durante o processo de planejamento, o projetista também tenta compreender como as diferentes classes estão relacionadas e como eles interagem para fornecer a funcionalidade desejada. Este aspecto da modelagem ajuda a construir os projetos que são mais propensos a satisfazer todos os requisitos do sistema. Devido à capacidade da UML para criar modelos diferentes, tornou-se uma ajuda para entender o sistema, projetar o sistema, bem como uma notação para a representar o projeto (JALOTE, 2005).

2.7 Considerações finais

O Python é uma linguagem de programação que tem uma ampla abrangência nas áreas da computação citadas. Ela mostrou-se uma tecnologia flexível e bastante completa para os interesses do projeto. Sua biblioteca traz um conjunto de módulos que auxiliam na resolução de diversos problemas computacionais, desta forma, proporcionando ao desenvolvedor uma estrutura segura e bem solida para criação de programas.

O *framework* Django reforça o poder da linguagem Python estendendo as suas funcionalidades para o ambiente da *Web*. Sua biblioteca ajuda o desenvolvedor a produzir com mais rapidez e qualidade, tornando-se uma tecnologia muito atraente para qualquer sorte de aplicativo voltado para *Web*.

O Python tem sido a escolha de grandes corporações, resumidamente, pode-se citar a ferramenta de busca do Google[6] e a pioneira de exploração espacial NASA[7]. Um estudo realizado pela Evans Data Corporation revela que o uso do Python cresceu, aproximadamente, 45% desde maio de 2008. Antes de março de 2008, apenas 13% dos desenvolvedores utilizavam a linguagem Python (VALLE, 2009).

Neste contexto, o Python e o Django formam, então, uma escolha adequada para a execução deste projeto, possibilitando atingir os objetivos iniciais propos-

[6]Disponível em: `http://code.google.com/intl/pt-BR/appengine/kb/general.html`
[7]Disponível em: `http://search.nasa.gov/search/search.jsp?nasaInclude=python`

tos. Através da sua modularidade, o Python permite ao desenvolvedor ampliar as funcionalidades do *software* proposto sem impactar profundamente no processo de desenvolvimento.

Os conceitos de redes de computadores e arquitetura cliente-servidor apresentados puderam determinar qual o meio de comunicação a ser utilizado na execução deste projeto. A comunicação de computadores através de aplicativos implementando *sockets* concorrentes orientados à conexão é a que se encaixa no propósito do *software*. Com este conceito, será possível realizar a comunicação entre os computadores em uma rede.

A linguagem Python contém boa parte dos conceitos de rede apresentados disponíveis em suas bibliotecas. Desta forma, o programador não necessita se preocupar com detalhes de mais baixo nível na construção de aplicativos que se comunicam através de uma rede. Para a arquitetura cliente-servidor em conjunto com o protocolo de comunicação TCP/IP, será adotado o módulo `socket`.

O XP é uma metodologia de desenvolvimento para pequenas equipes, algumas de suas práticas não podem ser utilizadas efetivamente quando um projeto é desenvolvido por apenas um desenvolvedor como é o caso do PyTM. Por conta disso, apenas alguns atividades do XP foram utilizadas. O núcleo do XP é formado por seus valores e princípios, as práticas e atividades se sustentam neste núcleo funcionando durante todo o ciclo de vida de desenvolvimento do *software*. Em termos de práticas e atividades, como o XP expressa suas práticas por meio das atividades, o desenvolvimento do PyTM fez uso principalmente de: liberações pequenas, metáfora, projeto simples, *refactory*, integração contínua e padrões de codificação (guia de estilo do Python[8]).

[8]Disponível em: `http://www.python.org/dev/peps/pep-0008`.

Capítulo 3

Proposta de desenvolvimento do PyTM

3.1 Considerações iniciais

Este capítulo tem como objetivo identificar os requisitos do *software* a ser desenvolvido com a linguagem Python para gerenciamento de tarefas. Baseado nos conceitos da análise orientada a objetos associados ao UML e XP, pretende-se identificar as partes que compõem o *software* bem como suas características para, com isso, promover o desenvolvimento de *software* propriamente dito.

Através do uso de recursos na notação UML, toda a modelagem lógica do sistema será representada permitindo a compreensão de como deverá ser o *software* final. Estes elementos ajudam no entendimento das funcionalidades que o *software* deve ter e é ponto de partida para a codificação deste.

Ainda neste capítulo, será abordado o processo de desenvolvimento, as ferramentas utilizadas para este fim, bem como uma explicação de cada módulo linguagem Python que foram necessários para a execução do projeto. O desenvolvimento da interface *Web* também será tema desta parte do texto.

Como se trata de um sistema utilizando o Python, optou-se por chamá-lo Python Tasks Manager e que também poderá ser referenciado pelo acrônimo PyTM.

3.2 Definição do PyTM

3.2.1 Requisitos do PyTM

De acordo com o projeto orientado a objetos, um sistema de *software* é decomposto em classes, que encapsulam dados e operações definindo novos tipos de dados abstratos, cujas operações são exportadas através da interface da classe. Para identificar estes objetos, inicialmente, deve-se especificar o que se espera do sistema.

No XP, existe o conceito de estórias de uso, que podem ser compreendidas como especificações de funcionalidades que se espera que o sistema possa efetuar, porém, não tem a intenção de ser um documento de requisitos completo. Para o desenvolvimento do PyTM, as estórias de uso são definidas a seguir:

1. O sistema deve possuir um módulo servidor que centralizará toda gerência dos computadores na rede, este módulo deve ser capaz de receber e registrar as informações de cada computador que se conecta a ele. Além disso, ele deve ser responsável por atribuir as tarefas para cada computador;

2. O sistema deve possuir um módulo cliente que atuará em cada computador da rede que se deseja gerenciar através dos *scripts*. Ele deve ser capaz de coletar e enviar informações de *hardware* e *software* para o módulo servidor e deste receber as tarefas para serem executadas;

3. Um administrador de sistemas deve ter a sua disposição uma interface *Web* para gerenciar uma rede de computadores visualizando os computadores nesta interface, escrevendo as tarefas em *scripts* e agendando-as conforme sua necessidade. Ele deve ter também a sua disposição algumas informações de *software* e *hardware* da cada computador.

A partir destas descrições, é possível identificar os componentes: módulo servidor, módulo cliente e administrador. Estes partes evidenciam-se como possíveis objetos que interagem entre si com seus atributos e ações. Para agregar mais informação ao processo de identificação utilizado no XP, serão associados os diagramas de caso de uso do UML.

3.2.2 Diagrama de componentes

No UML, um componente é uma parte encapsulada, reutilizável e substituível do *software*. Bons candidatos para componente são itens que executam uma funcionalidade essencial e serão usados, com frequência, em todo sistema (HAMILTON; MILES, 2006).

Um diagrama representa graficamente a arquitetura projetada, onde os componentes constroem entre si relações de dependência. A Figura 3.1 ilustra os componentes necessários para o funcionamento do projeto proposto. O componente Banco de Dados representa uma dependência tanto para Interface *Web* como para o módulo Servidor.

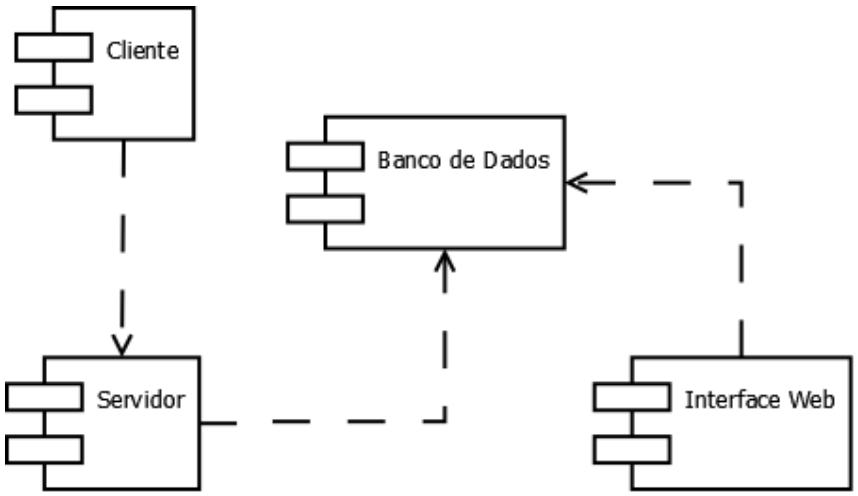

Figura 3.1: Diagrama de componentes.

3.2.3 Diagramas de caso de uso

Um caso de uso é um documento narrativo que descreve a sequência de eventos de um ator (um agente externo) que usa um sistema para completar um processo. Eles são histórias ou casos de utilização de um sistema. Casos de uso não são exatamente especificação de requisitos ou especificação funcional, mas ilustram e implicam requisitos na história que eles contam. Um diagrama de caso de uso ilustra um conjunto de casos de uso para um sistema, os atores, e a relação entre atores e os casos de uso (LARMAN, 2000).

Conforme a descrição de requisitos, os atores que irão compor o sistema são o usuário administrador do sistema, o módulo servidor e o módulo cliente. O administrador irá interagir com o módulo servidor através da interface *web* fazendo requisições e entrando com as tarefas. O módulo servidor irá se relacionar com o módulo cliente recebendo deste as informações do computador e enviando as tarefas a serem executadas.

A Figura 3.2 ilustra o ator Administrador do sistema e os casos de uso associados.

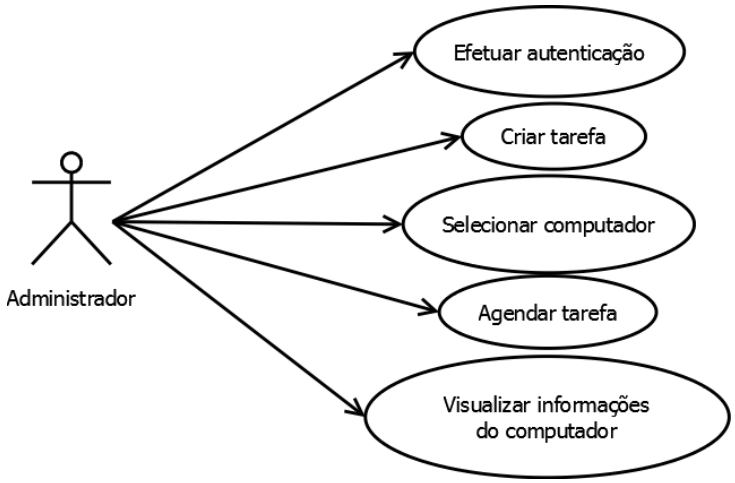

Figura 3.2: Diagrama de caso de uso do usuário Administrador.

Neste caso, o Administrador está interagindo com o PyTM nos seguintes casos:

- **Efetuar autenticação:** É o processo de identificação do usuário que tem acesso à interface do PyTM, uma fez devidamente identificado o usuário, terá acesso às funcionalidades disponíveis no sistema;
- **Criar nova tarefa:** Neste caso de uso, o Administrador irá informar ao sistema qual tarefa deverá ser realizada. Esta tarefa pode ser especificada através da informando um *script*;
- **Selecionar computador:** O Administrador irá escolher os computadores para submeter à tarefa criada;
- **Agendar tarefa:** Após a criação da tarefa e a seleção do computador, o administrador poderá optar pelo agendamento da execução da tarefa;
- **Visualizar informação do computador:** O Administrador pode acessar informações detalhadas de algum computador.

As ações do módulo Servidor são representadas no diagrama da Figura 3.3 e descritas a seguir:

- **Aceitar conexões:** O servidor deve ficar num estado de espera por conexões provenientes dos clientes;

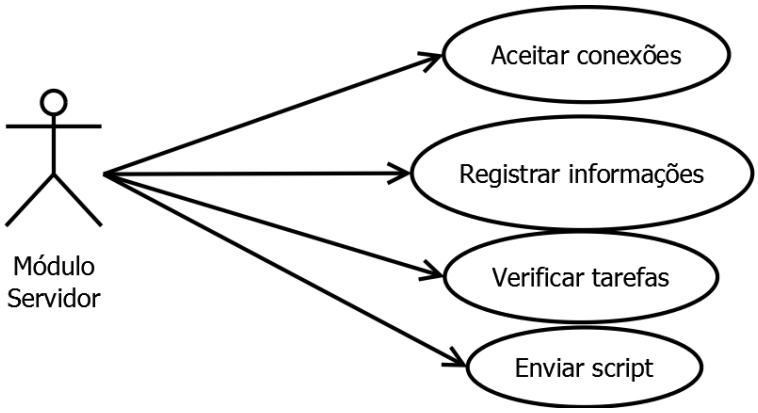

Figura 3.3: Diagrama de caso de uso do Módulo Servidor.

- **Registrar informações:** As informações que são enviadas pelos clientes devem persistir em algum sistema gerenciador de banco de dados;

- **Verificar tarefas:** Ao receber uma conexão de um computador cliente, o servidor deverá verificar a existência de alguma tarefa associada àquele computador;

- **Enviar *scripts*:** Caso existam tarefas definidas para um computador cliente, o servidor deverá enviá-la para execução.

Por fim, as ações do módulo Cliente irão descrever a forma que este atuará no sistema, a Figura 3.4 representa o diagrama de casos de uso, a saber:

- **Coletar informações do computador:** O módulo cliente deverá realizar rotinas que visam a buscar por informações sobre alguns componentes do computador a fim de formar um inventário de *hardware* e *software*. Estas informações serão organizadas para envio ao servidor;

- **Conectar ao servidor:** As ações do módulo cliente só terão sentido ao conectar com o módulo servidor, a partir desta ação, este módulo estará apto para realizar seu objetivo. A conexão será feita através dos conceitos de *socket*;

- **Enviar informações para o servidor:** Depois de conectado, o módulo cliente irá enviar as informações que coletou do computador;

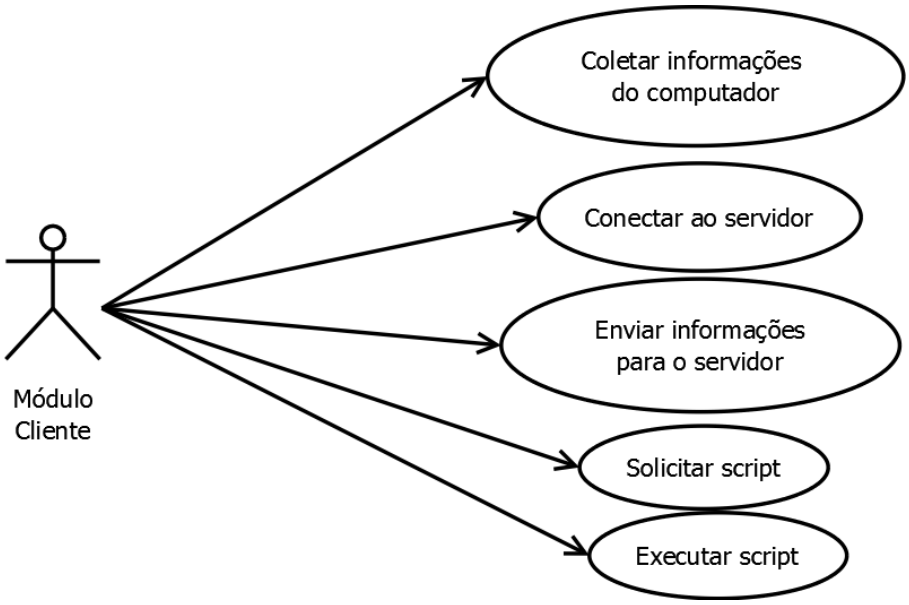

Figura 3.4: Diagrama de caso de uso do Módulo Cliente.

- **Solicitar *script*:** O módulo cliente só terá conhecimento das tarefas que deve executar através da solicitação ao módulo servidor que centraliza esta função;
- **Executar *script*:** Existindo uma tarefa para o computador, o módulo cliente será responsável por recebê-lo do servidor e executá-lo na forma de *script*.

Com as descrições feitas dos componentes do PyTM através dos casos de uso, será possível guiar o desenvolvimento do projeto, permitindo identificar classes e objetos, e a forma que eles atuarão.

3.2.4 Diagrama de implantação

Um diagrama de implantação mostra como e onde o sistema será implantado. Máquinas físicas e processadores são refletidos como nós, e a construção interna pode ser representada pela incorporação de nós ou artefatos. Artefatos modelam entidades físicas, como arquivos, *scripts*, tabelas de banco de dados, página *web*, arquivos JAR, e assim por diante, enquanto que nós modelam recursos computacionais, como computadores e drives de disco (ROSENBERG; STEPHENS, 2007).

A Figura 3.5 ilustra o diagrama de implantação, este diagrama permite melhor compreensão da disposição do PyTM no que diz respeito tanto ao módulo Servidor

quanto ao módulo Cliente. A depender da disponibilidade de acesso à Internet, a interface *Web* poderá estar disponível tanto na rede local como através da Internet.

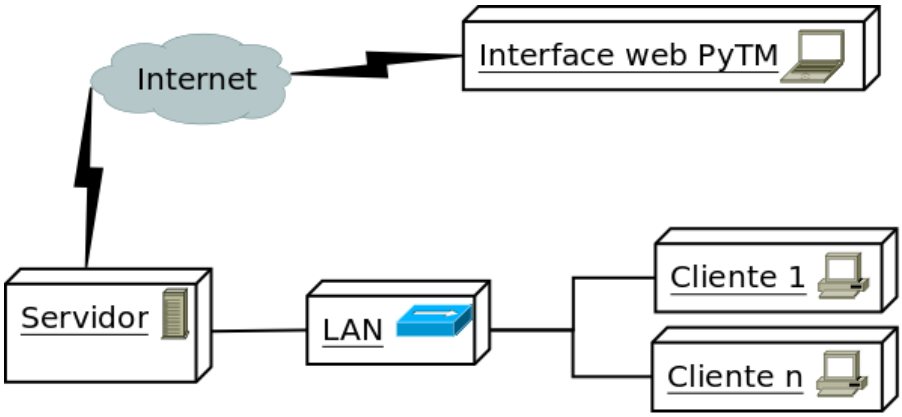

Figura 3.5: Diagrama de implantação.

3.3 Desenvolvimento do PyTM

Para a realização deste projeto, deve-se preparar um ambiente com os programas e ferramentas necessários para o objetivo de desenvolvimento de *software*. As distribuições GNU/Linux geralmente apresentam diversas opções para desenvolvimento de *software*, incluindo em seus repositórios IDEs (Integrated Development Environment), bibliotecas de desenvolvimento, depuradores, etc. Além destes programas estarem disponíveis para uso através de pacotes, também tem-se disponível seus códigos que servem como fonte de conhecimento, tudo isso seguindo a filosofia do *software* livre.

Não há aqui a intenção de mostrar a instalação de cada programa, porém pretende-se dar uma breve descrição do seu uso para a realização deste projeto.

3.3.1 Ambiente de desenvolvimento

O ambiente de desenvolvimento foi configurado sobre a distribuição GNU/-Linux Fedora versão 14. O projeto Fedora[1] foca na utilização de *software* livre e aberto, é mantido pela comunidade de desenvolvedores e patrocinado pela Red

[1]Disponível em: `http://fedoraproject.org`.

Hat[2]. Com o sistema operacional pronto, o conjunto de *software* instalado no ambiente de desenvolvimento foi:

- **Eclipse Platform[3] com PyDev[4]:** A plataforma Eclipse provê um conjunto de estruturas e serviços para formar uma plataforma de ferramenta universal. Esta é uma IDE que trata vários tipos de recursos (arquivos Java, C, C++, HTML, Python, etc.) de maneira genérica, mas não sabe o que fazer com cada tipo específico. O Eclipse faz uso de *plugins* que preparam a plataforma para trabalhar com estes diferentes tipos de recursos. Para trabalhar com recursos de projetos Python e Django, existe o plugin PyDev. O PyDev é uma IDE Python para o Eclipse que possui integração com o Django;

- **MySQL:** O MySQL Comunity Edition[5] é um sistema gerenciador de banco de dados de código aberto. Este recurso será responsável pela persistência das informações relevantes ao funcionamento do sistema. O MySQL foi escolhido devido a sua popularidade, suporte a diferentes plataformas além do Linux e por possuir diversos conectores disponíveis para muitas linguagens, inclusive o Python.

3.3.2 Implementação dos módulos Servidor e Cliente

O desenvolvimento dos módulos Servidor e Cliente do PyTM foram realizados utilizando diversos recursos disponíveis da linguagem de programação Python. Além das bibliotecas disponíveis dentro do ambiente de execução padrão do Python, existe o Python Package Index (PyPI)[6], que é um repositório de pacotes de terceiros.

Para o funcionamento adequado do *software*, alguns módulos Python foram comuns tanto no Servidor quanto no Cliente. Dentre todos os módulos, destacam-se:

- **cPickle[7]:** Este módulo implementa um algoritmo de serialização da estrutura de um objeto Python. Através dele, um objeto é transformado num fluxo de *bytes* (*pickling*) e pode ser enviado via rede do cliente para o servidor e pode ser transformado de volta num objeto (*unpickling*);

[2]Disponível em: `http://www.redhat.com`.
[3]Disponível em: `http://www.eclipse.org/platform`.
[4]Disponível em: `http://pydev.org`.
[5]Disponível em: `http://www.mysql.com/products/community`.
[6]Disponível em: `http://pypi.python.org/pypi`.
[7]Disponível em: `http://docs.python.org/library/pickle.html`.

- **socket**[8]**:** Este módulo representa uma tradução mais fácil das bibliotecas e chamadas de sistema UNIX para *sockets* através do estilo orientado a objetos do Python. É utilizado para enviar e receber os objetos serializados entre o Cliente e Servidor na aplicação PyTM;

- **daemon**[9]**:** Este módulo está disponível no repositório PyPI. Ele implementa as especificações para a execução de um daemon UNIX. Um daemon é um processo que fica executando continuamente em segundo plano sem intervenção direta. Este módulo foi utilizado para permitir que o PyTM fique executando de forma padronizada e adequada para ambientes de execução UNIX.

3.3.3 Módulo Servidor

O Módulo servidor ficará responsável por gerenciar as conexões com os clientes e a conexão com o banco de dados. Ele irá receber as informações enviadas pelos clientes e deverá registrá-las no banco de dados, como tem este papel centralizador, somente ele tem a necessidade de usar um módulo que permita conexão com o banco de dados.

Para realizar esta tarefa, optou-se por utilizar um meio que permitisse abstrair os detalhes da conexão com o banco bem como a necessidade do uso da linguagem própria desse para manipulação dos dados, conhecida como SQL.

No repositório PyPI, existe o módulo Elixir[10] que é uma camada de abstração para um banco de dados. Ele permite que objetos Python sejam diretamente mapeados para tabelas num banco de dados. Dessa forma, pode-se utilizar qualquer banco de dados que o Python suporte sem se preocupar com os detalhes de cada um.

A Figura 3.6 ilustra uma declaração de objeto representando um computador (classe Maquina) e algumas informações sobre esse. Antes da declaração, existe a *string* de conexão com o banco de dados (linha 5), logo após a declaração da classe (linhas 8 a 13), vêm seus campos com o tipo de dados que devem conter. Tudo isso será transformado em tabelas no banco pelo Elixir.

3.3.4 Módulo Cliente

O Módulo Cliente tem como principal objetivo executar as tarefas que lhe são atribuídas, adicionalmente, deve prover algumas informações sobre o com-

[8]Disponível em: `http://docs.python.org/library/socket.html`.
[9]Disponível em: `http://pypi.python.org/pypi/python-daemon`.
[10]Disponível em: `http://pypi.python.org/pypi/Elixir`.

```
1  #!/usr/bin/env python
2
3  from elixir import *
4
5  metadata.bind="mysql+mysqldb://root:mypass@localhost/db_pytm"
6  metadata.bind.echo = True
7
8  class Maquina(Entity):
9      mac = Field(Unicode(20))
10     ip = Field(Unicode(15))
11     nome = Field(Unicode(60))
12     so = Field(Unicode(20))
13     ativo = Field(Integer)
```

Figura 3.6: Exemplo de uso do módulo Python Elixir.

putador onde está sendo executado. Para cumprir os objetivos propostos, fez-se uso de módulos Python disponíveis em sua biblioteca padrão, como `platform`, `os` e `subprocess`. Foi necessário recorrer ao repositório PyPI para o módulo `pynetinfo`, a única exceção foi o módulo `python-dmidecode`, que não está no repositório PyPI. Todos eles são descritos a seguir:

- **platform**[11]**:** Este módulo Python fornece um conjunto de dados de identificação da plataforma subjacente. Estes dados dizem respeito a informações, como nome do computador, arquitetura do sistema operacional, nome e versão do sistema operacional, identificação do processador, etc.;

- **os**[12]**:** Este módulo provê um meio portável de acessar certas funcionalidades de cada sistema operacional. Com ele, é possível gerenciar arquivos e diretórios, criar e gerenciar processos, descritores de arquivos, etc.;

- **subprocess**[13]**:** Este módulo permite a criação de novos processos, conectar seus *pipes* de entrada, saída padrão e de erro, bem como obter os códigos de retorno de cada processo. Neste projeto, é utilizado para gerenciar a execução do *script* no cliente;

- **pynetinfo**[14]**:** Atualmente, o módulo tem a função de consultar e definir configurações das interfaces de rede em sistemas Linux apenas. Com ele, é pos-

[11] Disponível em: `http://docs.python.org/library/platform.html`.
[12] Disponível em: `http://docs.python.org/library/os.html`.
[13] Disponível em: `http://docs.python.org/library/subprocess.html`.
[14] Disponível em: `http://pypi.python.org/pypi/pynetinfo`.

sível listar, consultar e definir interfaces de rede, bem como listar e adicionar rotas de rede;

- **python-dmidecode**[15]**:** A DMI (Desktop Management Interface)[16] é uma estrutura padrão de gerenciamento de computadores, seu principal componente é uma base de dados com um conjunto de informações do computador e seus componentes de *hardware* que ficam armazenados na BIOS. O módulo `python-dmidecode` consulta as informações disponíveis em uma série de estruturas de dados da DMI e as disponibiliza em tipos de dados nativos do Python. Com este módulo, é possível, por exemplo, coletar informações sobre o modelo e fabricante de uma placa-mãe de um computador.

A Figura 3.7 ilustra um trecho de um programa em Python utilizando parte dos recursos dos módulos citados. Dentre os módulos utilizados, o dmidecode necessita de permissão de administrador, pois faz acesso a uma estrutura de dados protegida. Combinando adequadamente as informações fornecidas pelos módulos, é possível montar um relatório rico em informações sobre o sistema operacional e o *hardware* do computador. A Figura 3.8 exibe o resultado da execução deste programa.

```
1  #!/usr/bin/env python
2
3  import os, platform, dmidecode, netinfo, subprocess
4
5  print os.uname()[:3]
6  print platform.platform()
7  print subprocess.check_output(["/bin/uname", "-snr"])
8  print dmidecode.baseboard()['0x0002']['data']['Manufacturer']
9  print dmidecode.baseboard()['0x0002']['data']['Product Name']
10 print dmidecode.processor().values()[0]['data']['Version']
11
12 for dev in netinfo.list_active_devs()[1:]:
13     ip = netinfo.get_ip(dev)
14     mac = netinfo.get_hwaddr(dev)
15     break
16
17 print mac, ip
```

Figura 3.7: Exemplo de uso dos módulos Python para coleta de informação.

[15]Disponível em: `http://www.autonomy.net.au/pages/viewpage.action?pageId=1114783`.

[16]Disponível em: `http://www.dmtf.org/standards/dmi`.

```
[radson@sonata]$ sudo ./exemplo_mod_cliente.py
('Linux', 'sonata', '2.6.40.3-0.fc15.i686.PAE')
Linux-2.6.40.3-0.fc15.i686.PAE-i686-with-fedora-15-Lovelock
Linux sonata 2.6.40.3-0.fc15.i686.PAE

Dell Inc.
02K3Y4
Intel(R) Core(TM) i7 CPU       M 640  @ 2.80GH
58:94:6B:FC:74:14 192.168.1.4
```

Figura 3.8: Resultado da execução da coleta de informações.

3.3.5 Implementação da aplicação *Web*

A interface do usuário será feita através de uma aplicação *Web* desenvolvida com o *framework* Django. Este deve ser instalado no Fedora por meio dos seus repositórios ou da ferramenta de instalação que o próprio Python disponibiliza, chamada `setuptools`[17].

A Figura 3.9 mostra o ambiente de desenvolvimento do Eclipse com PyDev, no canto esquerdo da imagem, é ilustrado o espaço de trabalho com os projetos em uso, do lado direito, o editor de código. É importante notar a forma com que são organizados os diretórios e arquivos no projeto "pytmweb". O diretório "appweb" corresponde a uma aplicação dentro do projeto "pytmweb" que pode ter várias aplicações que compartilham configurações em comum. O arquivo exibido em questão é o que define as URLs de acesso às páginas das aplicações.

Dentro da aplicação "appweb", o recurso `models.py` irá conter a declaração dos objetos que serão mapeados para tabelas no banco de dados. A Figura 3.10 ilustra o objeto Maquina, que é uma representação lógica de um computador e suas características.

Já no recurso `views.py`, existem as funções que definem a lógica por trás de cada URL a ser requisitada no navegador. Geralmente, este recurso poderá importar as classes do `models.py` para poder realizar operações de consulta, inserção ou atualização no banco de dados. A Figura 3.11 ilustra as funções relativas à página principal da aplicação (*index*) e à página que permite adicionar uma nova tarefa (*addtask*).

[17] Disponível em: `http://pypi.python.org/pypi/setuptools`.

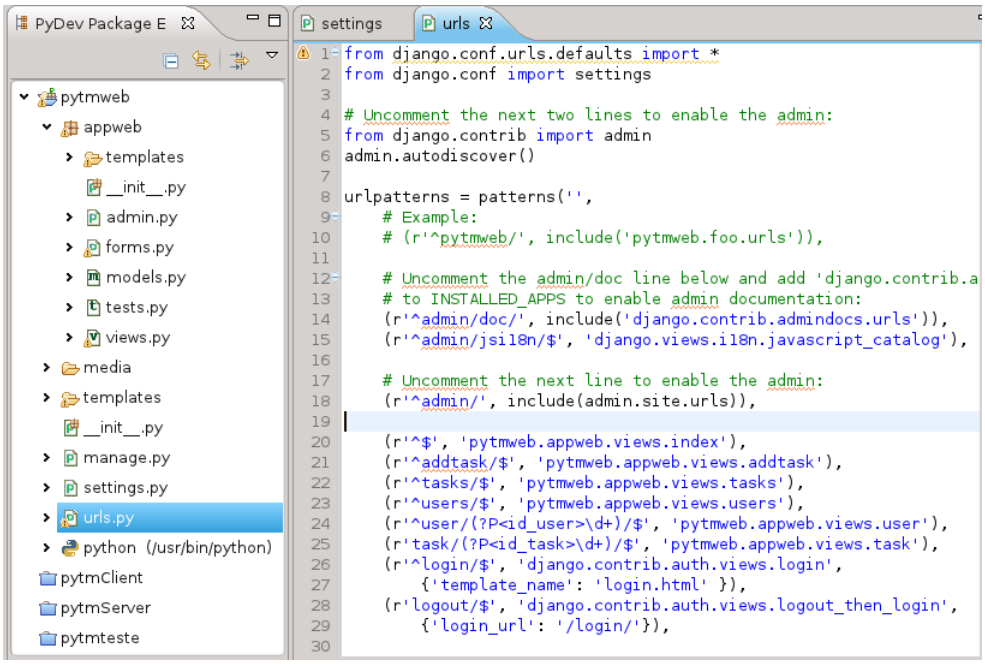

Figura 3.9: Codificando no Elipse com PyDev.

```
1  class Maquina(models.Model):
2      so = models.CharField(max_length=20)
3      mac = models.CharField(max_length=17, primary_key=True)
4      ip = models.CharField(max_length=15)
5      nome = models.CharField(max_length=20)
6      ativado = models.IntegerField()
7      conectado = models.IntegerField()
8      class Meta:
9          db_table = u'Maquina'
```

Figura 3.10: Trecho do `models.py` com declaração do objeto Maquina.

3.4 Considerações finais

Este capítulo foi destinado a uma especificação detalhada da estrutura do projeto PyTM. O uso da análise orientada a objetos e da metodologia de especificação dos requisitos utilizando UML permitiu alcançar a meta de identificação dos requisitos do projeto de forma mais clara e objetiva. Os diagramas do UML servem

```
1  @login_required
2  def index(request):
3      maquinas=Machine.objects.all()
4      return render_to_response("index.html", {'maquinas': maquinas},
5                                context_instance=RequestContext(request))
6
7  @login_required
8  def addtask(request):
9      if request.method == 'POST':
10         form = FormTask(request.POST)
11         if form.is_valid():
12             form.save()
13             return render_to_response("salvo.html", locals(),
14                                context_instance=RequestContext(request))
15     else:
16         form = FormTask()
17     return render_to_response('addtask.html', locals(),
18                                context_instance=RequestContext(request))
```

Figura 3.11: Trecho do `views.py` com declaração do `index` e `addtask`.

de fonte de informação para codificação dos recursos identificados e também serve como documentação do projeto.

A explanação sobre o ambiente de desenvolvimento do PyTM permitiu conhecer aplicações voltadas para a área de criação para *Web*. Os módulos Python forneceram os meios para concretização dos modelos idealizados, seus recursos e exemplos utilizados tornaram-se ponto de partida para construção dos módulos Servidor e Cliente.

Capítulo 4

Resultados e discussão

4.1 Considerações iniciais

Este capítulo tem o objetivo de apresentar os resultados obtidos com a execução deste projeto. Pretende-se examinar o produto final deste projeto, verificando quais benefícios foram realmente úteis para o administrador de sistemas. O uso dos conceitos até aqui abordados devem convergir para a resolução de um problema comum em ambientes de TI.

O resultado principal é o *software* Python Task Manager. Este produto deverá proporcionar ao administrador de sistemas uma abordagem pragmática na tarefa de gerenciar uma rede de computadores. Espera-se que o administrador possa ter maior controle sobre sua infraestrutura e de forma eficiente.

4.2 Interface de gerenciamento do PyTM

Durante o levantamento de requisitos, o módulo Servidor e módulo Cliente foram idealizados para prover um mecanismo de comunicação e automação do processo de execução de tarefas. Estes recursos devem realizar seu objetivo sem necessitar da intervenção de um agente externo, entenda-se, o administrador.

Para o administrador, fica disponível a interface de gerenciamento. Esta interface permitirá que sejam executadas atividades de submissão de tarefas e visualização de informações dos computadores da rede.

A Figura 4.1 ilustra a interface da página principal do PyTM, onde o administrador acessa uma tabela, exibindo a listagem de computadores, podendo selecionar algum dos itens da tabela para visualizar informações mais detalhadas. É possível também realizar busca de um registro desta tabela através de informações, como endereço IP ou nome do computador. Está página permite ao administrador

ter uma visão completa da rede gerenciada, pois pode visualizar todos os computadores que estejam executando o módulo Cliente do PyTM e ter controle sobre eles. É possível também selecionar um computador para exibir mais informações detalhadas sobre o sistema hospedeiro, conforme mostra a figura.

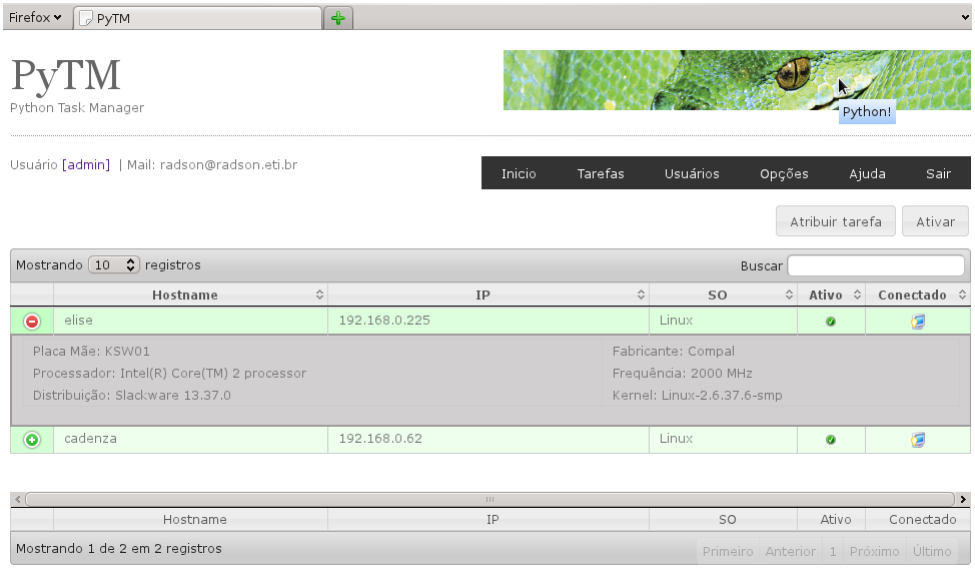

Figura 4.1: Página inicial do PyTM, listagem de computadores.

Para a criação de novas tarefas, existe um formulário no qual todas as informações relevantes a uma tarefa deverão ser informadas. Nesta página, o *script* poderá ser informado bem como dia e horário da execução. Com intuito de tornar a navegação agradável ao usuário da interface, fez-se uso de controles de formulário próprios da página de administração do Django. A Figura 4.2 exibe a página de criação de tarefas.

4.3 Impacto no gerenciamento de sistemas

A complexidade de gerenciar uma rede de computadores pode aumentar de acordo com a quantidade de equipamentos que se deve ter controle. Atividades comuns dizem respeito a manter o funcionamento adequado do sistema operacional, manutenção de contas de usuários, instalação de *software* e equipamentos, etc. Geralmente, estas atividades requerem a intervenção humana *in loco* ou de forma remota. Em ambos os casos, uma ou várias pessoas dedicam um tempo determinado para realizar a tarefa.

Figura 4.2: Página de edição de tarefas do PyTM.

Quando se trata de uma rede pequena com poucos computadores, o tempo gasto para execução de uma única tarefa não chega a representar uma perda de produtividade. Em casos de redes maiores com centenas ou milhares de computadores, o tempo dispensado para executar esta mesma tarefa poderá ser demasiado alto, prejudicando, assim, a produtividade dos recursos alocados para tal fim.

O PyTM atua especificamente neste problema da manutenção de computadores em uma rede, propondo o uso de *scripts* automatizados para realização das tarefas. Na mesma situação citada, em vez de um administrador acessar cada máquina para executar a tarefa ou, até mesmo, acessar cada computador remotamente, será apenas necessário criar um *script* que realize determinada tarefa e agendá-lo para a execução nos computadores.

Com o uso do PyTM, o resultado esperado é um menor esforço necessário para a realização de tarefas de gerenciamento, o administrador poderá se dedicar a outras atividades estratégicas para a gerência de TI. Além disso, o PyTM irá prover um conjunto de informações dos equipamentos que poderá servir para a tomada de decisões gerenciais. O PyTM foi, inicialmente, idealizado como uma ferramenta de auxílio ao administrador, porém pode ser também visto como um fomentador de um modelo de gerência de TI mais eficiente.

4.4 Objetivos alcançados

O desenvolvimento de *software* utilizando as metodologias propostas prevê como resultado um potencial produto ao final de cada ciclo. O PyTM chega a sua primeira versão atendendo aos objetivos elencados Capítulo 1 deste trabalho, com isso, tornando-se um mecanismo atraente para o administrador de sistemas.

O PyTM foi totalmente desenvolvido utilizando a linguagem Python, que se mostrou apta para resolução do problema computacional sem recorrer a outros mecanismos de apoio. O Python, por ser uma tecnologia portável, permitirá que o PyTM seja portado para outras plataformas que se encaixem no contexto proposto de gerência de computadores sem muito esforço.

Através da interface de *Web* de gerenciamento do PyTM, a manutenção dos computadores pode ser feita de forma remota, com a atuação do administrador submetendo tarefas automatizadas por meio dos *scripts* na interface *Web*. Além disso, o sistema irá permitir a coleta e exibição de informações detalhadas dos computadores.

Com a realização deste projeto, os objetivos traçados foram alcançados, a solução ainda poderá ser mais aprimorada com a possibilidade de agregar mais funcionalidades e recursos que forneçam ao administrador um sistema completo de gerenciamento.

4.5 Considerações finais

Após a realização deste trabalho, foi possível observar que os objetivos propostos e atendidos tornaram-se um marco inicial para a construção de um produto ainda mais elaborado. Todo o mecanismo desenvolvido tem potencial para trazer ainda mais resultados para os administradores de TI.

A linguagem de programação Python e seus recursos ainda têm muito mais a oferecer. O Python é uma tecnologia amplamente utilizada pela comunidade e por grandes corporações. Isso evidencia que haverá mais crescimento e melhorias na linguagem e tudo isso irá agregar ainda mais valor e funcionalidades ao PyTM.

O alvo deste projeto foi, desde o início, ser um facilitador para um administrador de TI. A cada dia, novas necessidades e desafios vão surgindo na área de gerencia de TI e, com o uso de tecnologias atualizadas, essas necessidades vão encontrando suas soluções.

O PyTM é uma solução inovadora, que usa tecnologia robusta e que é o primeiro passo para a criação de um produto ainda maior e mais completo. As comodidades fornecidas pelo Python e Django durante o processo de desenvolvimento promovem flexibilidade e versatilidade ao PyTM. A este produto poderão ser adi-

cionadas mais funcionalidades para dar suporte ao administrador de sistemas sem causar impactos profundos.

Capítulo 5

Considerações

5.1 Conclusões

A área de atuação a que se propôs este trabalho é ampla e sempre haverá espaço para novas soluções. Através da linguagem Python foi possível concretizar uma ferramenta de relevância para administração dos recursos de TI em uma rede de computadores. De forma flexível e escalável, o PyTM mostra-se como uma poderosa ferramenta para execução de tarefas remotas e com grande potencial para agregar mais funcionalidades.

A escolha do Python para a construção do PyTM foi adequada, devido à vasta quantidade de recursos da linguagem. Além disso, a facilidade de relacionamento do Python com as metodologias ágeis de desenvolvimento e com o paradigma de programação orientado a objetos, tudo isso proporcionou um desenvolvimento contínuo e progressivo.

5.2 Contribuições

O estudo sobre a linguagem de programação Python e seus recursos permitiu conhecer novos caminhos para criação de soluções em todas as áreas da computação. O Python tem características úteis para qualquer profissional na área de TI, desde administradores de rede, desenvolvedores de soluções científicas, comerciais, dentre outros.

Através deste trabalho, foi possível realizar um estudo mais profundo na forma em que aplicações se comunicam em uma rede de computadores. Tais princípios proporcionaram uma melhor compreensão do funcionamento básico da Internet e das diversas aplicações e serviços existentes.

Outra relevante contribuição foi referente à disciplina de engenharia de *software*. As metodologias ágeis de desenvolvimento trazem uma nova forma de trabalhar no processo de desenvolvimento de *software*. O foco no resultado reflete positivamente na rapidez e qualidade do *software* desenvolvido.

5.3 Trabalhos futuros

O *software* resultante deste projeto está focado na realização de tarefas. Adicionalmente, o PyTM coleta informações dos computadores armazenando-as em um banco de dados. Esta funcionalidade caracteriza, em parte, um CDBM (Configuration Data Base Management), que nada mais é do que um inventário dos equipamentos em uma rede. Esta funcionalidade é prevista pelo *framework* de governança ITIL.

Como trabalho futuro, pretende-se realizar um estudo mais profundo nas melhores práticas de gerenciamento de recursos de TI. A intenção é detectar as necessidades mais comuns aos administradores de recursos de TI e oferecer funcionalidades no PyTM que outras ferramentas não tenham alcançado. O objetivo é oferecer uma ferramenta completa através dos preceitos do *software* livre, já que grande parte das ferramentas pesquisadas são proprietárias.

Ainda como próximas atividades, está previsto adicionar mais recursos como: mecanismos de segurança, registro de eventos, o uso de ações pré-definidas e envio de comandos em tempo real. Além disso, pretende-se realizar testes em laboratório para atestar a melhora do processo de gerenciamento com o uso do PyTM.

Com intuito de fomentar os ideais do movimento Software Livre, este projeto encontra-se disponível na internet. O PyTM está armazenado no banco de dados de projetos de código aberto conhecido como SourceForge.net[1]. Este provedor fornece hospedagem dos arquivos de muitos projetos de código aberto, além disso fornece ferramentas para desenvolvimento colaborativo. O código do PyTM está disponível no endereço: `http://pytaskmanager.sf.net`.

[1] Disponível em: `http://sourceforge.net`.

Referências Bibliográficas

BENNETT, J. *Practical Django Projects*. 2. ed. USA: Apress, 2009. ISBN 978-1-4302-1938-5.

BORGES, L. E. *Python para Desenvolvedores*. Edição do autor. Rio de Janeiro - RJ: [s.n.], 2010. ISBN 978-85-909451-1-6.

BRANDAO, J. M. N. *Aprendendo Django no Planeta Terra*. 1. ed. [s.n.], 2009. Disponível em: <http://www.aprendendodjango.com>.

CHUN, W. J. *Core Python Programming*. 2. ed. [S.l.: s.n.], 2006. ISBN 0-13-226993-7.

CORREIA, C. H.; TAFNER, M. A. *Análise Orientada a Objetos*. 2. ed. Florianópolis - SC: Visual Books, 2006. ISBN 85-7502-200-8.

Django Project. *Django | FAQ: General | Django Documentation*. [S.l.], jun. 2010. Disponível em: <http://docs.djangoproject.com/en/dev/faq/general>.

FOROUZAN, B. A. *Comunicação de dados e redes de computadores*. 3. ed. Porto Alegre - RS: Bookman, 2006. ISBN 978-85-363-0614-8.

GIFT, N.; JONES, J. M. *Python for Unix and Linux System Administration*. 1. ed. [S.l.: s.n.], 2008.

GOERZEN, J. *Foundations of Python Network Programming*. [S.l.]: Apress, 2004. ISBN 1-59059-371-5.

HAMILTON, K.; MILES, R. *Learning UML 2.0*. 1. ed. USA: O'Reilly, 2006. ISBN 0-596-00982-8.

HOLOVATY, A.; KAPLAN-MOSS, J. *The Definitive Guide to Django: Web Development Done Right*. 2. ed. United States of America: Apress, 2009. ISBN 978-1-4302-1936-1.

HOURIEH, A. *Django 1.0 Web Site Development*. Birmingham, United Kingdom: Packt Publishing, 2009. ISBN 978-1-847196-78-1.

JALOTE, P. *An Integrated Approach to Software Engineering*. 3. ed. Department of Computer Science and Engineering - Indian Institute of Tecnology. Kanpur, India: Springer Science+Business Media, Inc., 2005.

LARMAN, C. *Utilizando UML e padrões: uma introdução à análise e ao projeto orientado a objetos*. Porto Alegre: Bookman, 2000. ISBN 85-7307-651-8.

LUCIA, A. D.; FERRUCCI, F.; TORTORA, G.; TUCCI, M. *Emerging methods, technologies, and process managemente in software engineering*. New Jersey: John Wiley & Sons, INC., 2008. ISBN 978-0-470-08571-4.

LUTZ, M. *Learning Python, Fourth Edition*. 4. ed. [S.l.]: O'Reilly Media, Inc., 2009. ISBN 978-0-596-15806-4.

NOVELL. *Novell ZENworks Configuration Management*. [S.l.], jan. 2010. Disponível em: <http://www.novell.com/documentation/zlm72%-/lm7admin/?page=/documentation/zlm72/lm7admin/data/bvb2jzt.html>.

PILGRIM, M. *Dive Into Python*. [S.l.]: Apress, 2004. ISBN 978-1590593561.

Portal do Software Público Brasileiro. *Portal do Software Público Brasileiro*. [S.l.], jan. 2010. Disponível em: <http://www.softwarepublico.gov.br/Oinequ e_e_o_SPB>.

Python Software Foundation. *Python Programming Language – Official Website*. [S.l.], jan. 2010. Disponível em: <http://www.python.org>.

PYTHONBRASIL. *Python Brasil*. [S.l.], jan. 2010. Disponível em: <http://www.python.org.br>.

REIS, C. *Caracterização de um Modelo de Processo para Projetos de Software Livre*. Disserta (Mestrado) — Instituto de Ciências Matemática e Computação - São Carlos - SP, http://www.async.com.br/ kiko/quali/, abr. 2001.

ROSENBERG, D.; STEPHENS, M. *Use Case Driven Object Modeling with UML: Theory and Practice*. USA: Apress, 2007. ISBN 978-1-59059-774-3.

SCHACH, S. R. *Classical and Object-Oriented Software Engineering*. 6. ed. [S.l.]: McGraw-Hill, 2004. ISBN 978-0072865516.

SEBESTA, R. W. *Concepts of Programming Languages*. 7. ed. [S.l.]: Pearson Education, 2006. ISBN 0-321-312511.

TANENBAUM, A. S. *Redes de computadores*. 4. ed. [S.l.: s.n.], 2003. ISBN 85-352-1185-3.

VALLE, J. D. Python está em alta, aponta estudo. *INFO Online*, 17 nov. 2009. Disponível em: <http://info.abril.com.br/noticias/ti/python-esta-em-alta-aponta-estudo-17112009-2.shl>.

YOUNKER, J. *Foundations of Agile Python Development*. United States of America: Apress, 2008. ISBN 978-1-4302-0636-1.