CÉSAR HENRIQUE KALLAS

CONECTA
GERENCIADOR DE COMPUTADORES EM REDE

PUC-CAMPINAS
2006

CÉSAR HENRIQUE KALLAS

CONECTA
GERENCIADOR DE COMPUTADORES EM REDE

Monografia apresentada como exigência da disciplina Projeto Final II, ministrada no Curso de Engenharia de Computação, na Pontifícia Universidade Católica de Campinas.

Orientador: Prof. Dr. Fernando Ernesto Kintschner
Co-orientador: Prof. Ms. Edmar Rezende

PUC-CAMPINAS
2006

BANCA EXAMINADORA

Presidente e Orientador Prof. Dr. Fernando Ernesto Kintschner
1º Examinador Prof. Dr. Carlos Miguel Tobar Toledo
2º Examinador Prof. Ms. Edmar Rezende

Campinas, 07 de Dezembro de 2006.

Aos meus pais Geraldo Majela e Sônia,
pelo exemplo de vida, amor
e por tornar meus sonhos uma realidade,
À minha namorada Karyna,
pelo companherismo, amor e força.

AGRADECIMENTOS

À toda minha família, pela ajuda e força ao longo deste caminho.

Ao co-orientador e Prof. Ms. Edmar Rezende e aos orientadores Prof. Dr. Fernando Ernesto Kintschner e Prof. Dr. Carlos Miguel Tobar Toledo, pela colaboração no desenvolvimento, pela atenção e dedicação.

A todos os professores da Pontifícia Universidade Católica de Campinas, pelo incentivo, estímulo e pela dedicação de ensinar.

À comunidade de *software* livre: desenvolvedores, grupos e usuários pelo suporte, programas, dicas e principalmente pelo conhecimento compartilhado.

The Zen of Python

Bonito é melhor que feio.
Explícito é melhor que implícito.
Simples é melhor que complexo.
Complexo é melhor que complicado.
Plano é melhor que aglomerado.
Esparso é melhor que denso.
Legibilidade faz diferença.
Casos especiais não são especiais o bastante para quebrar as regras.
Embora a praticidade vença a pureza.
Erros nunca devem passar silenciosamente.
A menos que sejam explicitamente silenciados.
Diante da ambigüidade, recuse a tentação de adivinhar.
Deve haver um -- e preferencialmente só um -- modo óbvio para fazer algo.
Embora esse modo possa não ser óbvio à primeira vista a menos que você seja holandês.
Agora é melhor que nunca.
Embora nunca freqüentemente seja melhor que *exatamente* agora.
Se a implementação é difícil de explicar, é uma má idéia.
Se a implementação é fácil de explicar, pode ser uma boa idéia.
Namespaces são uma grande idéia -- vamos fazer mais dessas!

por Tim Peters

RESUMO

KALLAS, César Henrique. Conecta – Gerenciador de Computadores em Rede. Campinas, 2006. 53f . Monografia (Graduação) – Curso de Graduação em Engenharia de Computação, Pontifícia Universidade Católica. Campinas, 2006.

Atualmente empresas e instituições gastam muito tempo e dinheiro com a manutenção do ambiente de programas nos computadores aos quais os usuários têm acesso. Mesmo treinando profissionais, ditando regras e padrões, os computadores em um mesmo ambiente quase sempre são mantidos (configurados e instalados) de maneira diferente. Uma das formas de melhorar o processo de manutenção dos computadores é automatizá-lo, podendo todo o processo ser feito por um número reduzido de profissionais, minimizando a possibilidade de haver desvios na execução da manutenção. Em resposta a esse problema, é proposto o desenvolvimento de uma ferramenta que gerencie esse processo de manutenção, através de um arquitetura que permita o gerenciamento dos computadores a partir de um único lugar, sem a necessidade de deslocamento físico do técnico. Com essa ferramenta, é possível realizar ações de instalação, atualização e remoção de programas nos computadores, rever todo o histórico de ações realizadas neles e ter acesso aos componentes de *software* e *hardware* (inventário) instalados em cada computador. O objetivo principal dessa automatização é reduzir o tempo gasto com a manutenção manual dos computadores. A ferramenta protótipo foi desenvolvida na linguagem de programação Python, com bancos de dados MySQL e SQLite, usando o sistema de pacotes RPM e conceitos de serviços HTTP e RPC. A avaliação e validação da ferramenta foram feitas através de tarefas pré-determinadas, cuja duração, com a ferramenta e sem ela, foi comparada. O resultado final da ferramenta é surpreendentemente melhor comparado à manutenção manual.

Termos de indexação: Manutenção de computadores, Gerenciamento de computadores, Instalação automática de programas, Inventário automático de computadores.

ABSTRACT

KALLAS, César Henrique. Conecta – *Networked Computers Manager*. Campinas, 2006. 53f . Monografia (*Graduation*) – Engenharia de Computação, Pontifícia Universidade Católica. Campinas, 2006.

Currently companies and institutions spend too much time and money with the maintenance of the program environment in computers to which users have access. Even training professionals, dictating rules and standards, computers in the same environment are almost always kept (configured and installed) in different ways. One of the forms to improve the process of computers maintenance is to automatize it, being able to do all the process by a reduced number of professionals and without the possibility of deviations in the maintenance execution. In reply to this problem, the development of a tool is considered to manages this maintenance process, through an architecture that allows the computers management from one place, without the necessity of physical locomotion of the technician. With this tool it is possible to carry out actions for installation, update, and removal of programs in the computers, to review the description of actions carried out on them, and to have access to the components of software and hardware (inventory) installed in each computer. The main objective of this automatization is to reduce the spent time related to the manual maintenance of computers. The tool prototype was developed with the programming language Python, with the data bases MySQL and SQLite, having used the system of RPM packages and concepts of services HTTP and RPC. The assessmet and validation of the tool were made through pre established tasks, whose duration time was compared when perfomed with and with out the tool. The final result of the tool is surpreendent better if compared to the manual maintenance.

Index terms: computer maintenance, computer management, automatic program instalation, automatic computer inventory.

LISTA DE FIGURAS

Figura 1: Diagrama de Arquitetura.....27
Figura 2: Arquitetura cliente e servidor simplificada.....28
Figura 3: Arquitetura cliente/servidor com os principais módulos.....28
Figura 4: Interface gráfica do ConectaGerente.....30
Figura 5: Gerenciamento do repositório de programas no ConectaGerente.....32
Figura 6: Interface gráfica do ConectaAgente.....34
Figura 7: Modelo Entidade Relacionamento - Banco de dados do Gerente.....36
Figura 8: Modelo Entidade Relacionamento - Banco de dados do Agente.....37
Figura 9: Programas selecionados para manutenção.....40
Figura 10: Comparação do tempo gasto (s) para a instalação manual e automatizada....44
Figura 11: Comparação do tempo gasto (s) para a atualização manual e automatizada. .45
Figura 12: Comparação do tempo gasto (s) para a desinstalação manual e automatizada
.....45
Figura 13: Tempo gasto de manutenção para executar as tarefas de instalação,
atualização e desinstalação.....46

LISTA DE TABELAS E QUADROS

Quadro 1: Exemplo de programação em Python (cálculo de fatorial).....21
Quadro 2: Exemplo de arquivo estruturado em XML.....31
Quadro 3: Exemplo da saída do comando TOP.....40
Tabela 1: Tempo gasto para a instalação de programas manualmente.....41
Tabela 2: Tempo gasto para a atualização de programas manualmente.....42
Tabela 3: Tempo gasto para a desinstalação de programas manualmente.....42
Tabela 4: Tempo gasto para a instalação de programas pelo protótipo do projeto.....43
Tabela 5: Tempo gasto para a atualização de programas pelo protótipo do projeto.....43
Tabela 6: Tempo gasto para a desinstalação de programas pelo protótipo do projeto.....43
Tabela 7: Comparação do tempo médio gasto para executar ações de forma manual e usando o protótipo proposto pelo projeto.....44

DEFINIÇÕES E ABREVIATURAS

SMART	=	*Smart Package Manager*
YOU	=	*Yast online update*
NTP	=	*Network Time Protocol*
KDE	=	K Desktop Environment
QT	=	*Toolkit Multiplatform GUI & aplication Development*
Python	=	Linguagem de programação orientada a objetos
Firefox	=	Navegador Web
SUSE	=	Distribuição Linux
Linux	=	Sistema Operacional
PUC Campinas	=	Pontifícia Universidade Católica
XML	=	*eXtensible Markup Language*
RPC	=	*Remote Procedure Call*
HTTP	=	*Hyper Text Transfer Protocol*
RPM	=	*Red Hat Package Manager*
TCP	=	*Transmission Control Protocol*
IP	=	*Internet Protocol*

SUMÁRIO

1 INTRODUÇÃO.....14
1.1 Delimitação do Problema.....15
1.2 Objetivo.....16
1.3 Estado da Arte.....16
1.4 Estrutura da Monografia.....17

2 FERRAMENTAS E TECNOLOGIA.....18
2.1 Plataforma e Sistema Operacional.....19
2.2 Banco de Dados.....19
2.3 Linguagem de Programação.....20
2.4 Interface de Desenvolvimento.....21
2.5 Interface Gráfica.....22
2.6 Sistema de Pacotes.....22
2.7 Estruturação dos Dados.....22
2.8 Chamada de Procedimento Remoto.....23

3 DESENVOLVIMENTO DO PROJETO.....24
3.1 Recursos Materiais.....25
3.2 Metodologia.....25
3.3 Arquitetura.....26
3.4 Funcionamento do Sistema.....26
3.5 Modelo Entidade Relacionamento do Banco de Dados.....34
3.5.1 Gerente.....35
3.5.2 Agente.....36

4 RESULTADOS.....38
4.1 Avaliação.....39
4.2 Validação.....41

5 CONCLUSÃO.....47
5.1 Dificuldades.....49
5.2 Limitações do Programa Protótipo.....49

5.3 Sugestões de Implementação.....50

6 REFERÊNCIAS.....51

1 INTRODUÇÃO

Toda empresa ou instituição, que possui uma rede de computadores, já percebeu ou perceberá que o mais difícil não é manter os servidores em funcionamento, mas sim manter os computadores aos quais os usuários têm acesso. Isso porque os servidores normalmente permanecem em uma sala fechada e com controle de acesso, enquanto os computadores dos usuários estão sujeitos ao uso diversificado e às suas curiosidades.

Particularmente, nas empresas aonde a disponibilidade e velocidade dos serviços é um fator de lucro, os computadores, tanto os servidores como os dos usuários, devem estar sempre prontos para uso, contando com um grupo de programas padrão e configurações.

Apesar dos profissionais de informática passarem por treinamentos, das empresas ditarem instruções de trabalho e padrões, os computadores em um mesmo ambiente quase sempre são configurados e instalados de maneira diferente, pois cada profissional possui um conhecimento e opera de maneira diferente ao prestar um suporte, nem sempre seguindo totalmente os padrões determinados, criando um ambiente computacional heterogêneo.

1.1 Delimitação do Problema

A manutenção de muitos computadores demanda tempo, profissionais, e padrões, para tornar o ambiente igual e disponível a todos os usuários.

Apesar dos investimentos, a manutenção tende a ter um alto custo, demorar muito, ser repetitiva e apresentar soluções diversas para um mesmo problema. Isso porque cada empresa, profissional ou equipe de suporte possui conhecimento e técnicas diferentes, tornando o serviço artesanal e dificultando a sua padronização.

Devido a isso, torna-se necessário automatizar o processo de manutenção dos computadores que estão em um mesmo ambiente (rede), podendo ser uma empresa, uma instituição de ensino ou qualquer ambiente que necessite.

Automatizando o processo de manutenção, as tarefas passam a ser realizadas de maneira padronizada, aumentando a disponibilidade e estabilidade dos computadores, acoplado a uma redução de custos, tempo e profissionais.

1.2 Objetivo

Tem-se como objetivo diminuir o tempo gasto para manter vários computadores em rede, com o uso de um protótipo que gerencie a manutenção dos computadores.

O programa protótipo deve automatizar as tarefas mais repetitivas de manutenção:

- Instalação, atualização e remoção de programas.

Essas tarefas se repetem porque há um número expressivo de *softwares* disponíveis, que preferencialmente devem sempre estar atualizados, para poder inibir problemas de segurança, corrigir falhas e prover novas funcionalidades.

A necessidade por instalação, atualização e desinstalação de *softwares* é quase freqüente nas organizações.

1.3 Estado da Arte

Atualmente não se conhece programa algum difundido de *software* livre que tenha o mesmo ideal de automatizar o processo de manutenção de computadores. Porém, existem projetos que fazem parte do que está sendo proposto, como a atualização de segurança (YOU, 2006), adicionar, remover e atualizar *softwares* manualmente (SMART, 2006).

O Smart é um gerenciador de pacotes que instala, remove e atualiza *softwares*, de acordo com comandos executados localmente em um computador, mas ele não executa automaticamente ações remotas e exige a interferência de uma pessoa. Se for preciso instalar um programa em um computador, é

necessário ir até o computador e executar o comando de instalação do programa.

Com o protótipo proposto, batizado de Conecta, será possível comandar a atualização de correções de segurança e também comandar a atualização, instalação e remoção de programas nos clientes.

1.4 Estrutura da Monografia

Nesse documento são tratados os problemas atuais e a solução que foi realizada através de um projeto de desenvolvimento para uma ferramenta que gerencia computadores em rede.

No capítulo 2, são abordadas as tecnologias utilizadas no desenvolvimento do projeto, incluindo a linguagem de programação, *hardware* necessário e as bases de dados utilizadas.

No capítulo 3, é descrito o desenvolvimento do protótipo do projeto, a metodologia de desenvolvimento, o seu funcionamento e o diagrama de arquitetura.

No capítulo 4, é feito o estudo dos resultados do projeto, a avaliação e validação dos dados obtidos com ele, afim de comprovar o objetivo especificado.

No capítulo 5, é feita a conclusão do projeto, debatendo sobre os resultados, a implementação, dificuldades e limitações da ferramenta.

2 FERRAMENTAS E TECNOLOGIA

Nesse capítulo são abordadas as tecnologias utilizadas no desenvolvimento do projeto, incluindo a linguagem de programação, base dados, bibliotecas, *hardware* necessário e as bases de dados utilizadas.

2.1 Plataforma e Sistema Operacional

Todo o *software* foi desenvolvido para a plataforma Intel x86, por ser a plataforma de hardware mais difundida.

O Sistema Operacional escolhido foi o OpenSuse Linux 10.1 (SUSE, 2006). A escolha do Linux como sistema operacional, deve-se ao fato de ser um *software* livre, ser extremamente estável, seguro e possuir uma imensa variedade e disponibilidade de ferramentas úteis para o desenvolvimento deste projeto.

2.2 Banco de Dados

A arquitetura do projeto, definida no Capítulo 3, exige duas bases de dados distintas, uma utilizada pelo servidor e outra pelo cliente.

O programa de banco de dados adotado do servidor é o MySQL (MYSQL, 2006), pois é um programa de fácil acesso, grande aceitação no mercado, fácil utilização e bom desempenho.

O MySQL provê uma ferramenta de gerenciamento de base de dados chamada MySQL *Administrator*. Essa ferramenta possui uma interface amigável para o acesso aos dados do banco de dados MySQL, com opções de gerenciamento da base de dados, incluindo estatísticas de tamanho e uso da base de dados, facilitando qualquer alteração direta nos dados.

O programa de banco de dados adotado no cliente é o SQLite (SQLITE, 2006), que provê duas características importantes: fácil acesso e é uma base de dados embarcada.

O fato do SQLite ser uma base de dados embarcada ajuda muito

tecnicamente, pois a base de dados pode ser distribuída junto com o *software* cliente, sem a necessidade de instalação e configuração dessa base de dados separadamente, o que favorece o objetivo deste projeto.

Toda a estruturação das bases de dados do servidor e do cliente é desenvolvida com o DbDesigner (DBDESIGNER, 2006). Com o DBDesigner é possível estruturar toda a base de dados usando o modelo de entidade e relacionamento (MER), o que facilita o desenvolvimento e possibilita visualizar de maneira gráfica a base de dados, tendo a possibilidade de exportar toda essa estrutura desenhada para a linguagem de base dados SQL, poupando tempo no desenvolvimento das bases de dados.

2.3 Linguagem de Programação

A linguagem de programação Python (PYTHON, 2006) foi utilizada no desenvolvimento de todo o sistema.

O Python foi criado em 1991 por Guido Van Rossum com o intuito de facilitar a expressão de idéias em código fonte, estando disponível desde então como *software* livre na Internet, para uso e melhoramentos no seu código.

O Python é uma linguagem interpretada, multi-plataforma, funciona em quase todos os sistemas operacionais disponíveis atualmente como: Linux, Microsoft Windows, Unix e MacOS.

O Python possui uma sintaxe clara e objetiva, orientação a objetos, tipagem forte (cada objeto tem um tipo definido e imutável) e tipagem dinâmica (não é necessário declarar o tipo da variável, basta apenas atribuir valor a ela, que o interpretador estabelece o tipo).

O Python inclui estruturas de dados úteis, tais como: listas, dicionários e uma variedade enorme de módulos (bibliotecas) para tratamento de arquivos XML (XML, 2006), assinaturas MD5, RPC (XMLRPC, 2006), conexão com banco de dados MySQL e SQLite, interface para gerenciamento de pacotes RPM (RPM, 2006) e a possibilidade de invocar comandos ao sistema operacional hospedeiro.

O código fonte produzido em Python é interpretado por uma máquina virtual, não precisando de nenhum processo de compilação por parte do usuário. Essa máquina virtual provê uma ótima interface de programação em tempo real, um ambiente interativo em que é possível codificar e obter resultados em tempo de programação (Quadro 1), podendo testar códigos fontes sem precisar editar nenhum arquivo diretamente. Este tipo de ambiente trouxe uma enorme vantagem ao projeto, como o poder de prototipar funções antes de inserí-las diretamente no código fonte do projeto.

Quadro 1: Exemplo de programação em Python (cálculo de fatorial)

```
[cesarkallas@cesarkallas ~]$ python
Python 2.4.2 (#1, May  2 2006, 08:13:46)
[GCC 4.1.0 (SUSE Linux)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> def fatorial(numero):
...         f = 1
...         while(numero > 0):
...             f = f * numero
...             numero = numero - 1
...         return f
...
>>> print fatorial(10)
3628800
```

Essas características do Python tornaram-se as grandes vantagens de utilizá-lo neste projeto, resultando em um código fonte de fácil leitura e em uma grande produtividade no desenvolvimento.

2.4 Interface de Desenvolvimento

Para a codificação do projeto foi utilizado o programa SciTE (SCITE, 2006).

A escolha do SciTE deve-se ao fato de ser desenvolvido para facilitar a programação em uma enorme variedade de linguagens, com um ambiente simplificado, interface amigável, além de estar disponível livremente (incluindo o código fonte) na Internet.

2.5 Interface Gráfica

Para a interface gráfica do projeto com o usuário foi utilizada a biblioteca gráfica QT (TROLLTECH, 2006).

O fator decisivo para escolha da QT para o projeto foi a facilidade de construir interfaces gráficas utilizando-se o programa QT *Designer*, e o curto espaço de tempo necessário para construí-las.

Apesar do QT Designer gerar apenas código fonte em C++, é possível utilizar as interfaces por ele geradas atráves do módulo pyQT (PYQT, 2006), que transforma esse código gerado pelo QT Designer em Python.

2.6 Sistema de Pacotes

O sistema de pacotes de programas escolhido no desenvolvimento do projeto foi o RPM (RPM, 2006).

A vantagem de usar esse sistema, de gerenciamento de pacotes de programas é que o OpenSuse o adota e o Python já possui módulo para comunicar e executar ações com eles.

2.7 Estruturação dos Dados

A linguagem *XML* (XML, 2006) foi utilizada no projeto para facilitar a troca de dados entre o servidor e o *software* cliente.

Essa linguagem possibilita a estruturação dos dados a serem trocados entre o servidor e o cliente, através de uma sintaxe clara e objetiva, proporcionando assim um sistema de troca de mensagens independente de sistema operacional, plataforma e linguagem, por se tratar de um formato de texto puro.

2.8 Chamada de Procedimento Remoto

O sistema de chamada de procedimento remoto permite que os clientes possam chamar procedimentos localizados no servidor, utilizando os recursos do próprio servidor.

Toda comunicação é feita por troca de mensagens, utilizando o protocolo TCP/IP através de um canal de comunicação (*socket*). Porém, essa troca de mensagens é intrínseca no módulo de desenvolvimento (transparente ao programador).

Usando este sistema, foi possível desenvolver uma solução de troca de mensagens entre o cliente e o servidor em um curto período de tempo.

3 DESENVOLVIMENTO DO PROJETO

Nesse capítulo é descrito o desenvolvimento do protótipo do projeto, a metodologia de desenvolvimento, o funcionamento do protótipo e o diagrama de arquitetura.

3.1 Recursos Materiais

Os recursos materiais utilizados na realização do projeto foram dois computadores ligados em rede, com a seguinte configuração:

Servidor:

- Processador baseado no Intel de 1800 Mhz.
- 512 MB de memória ram.
- Disco rídigo de 60 GB.
- Placa de rede *ethernet* 10/100 Mb.

Cliente:

- Processador baseado no Intel de 450 Mhz.
- 128 MB de memória ram.
- Disco rídigo de 5 GB.
- Placa de rede *ethernet* 10/100 Mb.

3.2 Metodologia

A metodologia adotada nesse projeto foi a Prototipação (MARCORATTI, 2006).

O modelo da metodologia Prototipação propõe um ciclo de desenvolvimento mais rápido e de alta qualidade.

A vantagem desse método, para este projeto foi a criação de protótipos, já com uma noção do que se pode esperar. Toda a análise de requisitos e funcionalidades do programas partem do próprio projetista e programador, diferente de outras metodologias que pretendem entender o que um

cliente externo precisa.

Esse método também possibilitou um desenvolvimento mais ágil, no curto espaço de tempo necessário para implementá-lo, visto que não é necessário criar documentações além dos próprios protótipos.

3.3 Arquitetura

A arquitetura do projeto é baseada no paradigma cliente e servidor (Figura 1). Essa arquitetura possui duas partes distintas: a arquitetura do servidor e a arquitetura do cliente, detalhadas a seguir.

Esse tipo de arquitetura foi escolhido pela necessidade de centralizar as informações dos clientes, haver uma escabilidade no número de clientes (facilidade de expansão) e transparência de localização dos clientes, tornando o serviço gerenciável e confiável.

Essa arquitetura também pôde prover um menor risco a falhas, porque permite replicar o servidor e expandir o serviço, acrescentando-se novos servidores.

3.4 Funcionamento do Sistema

Para o funcionando do sistema existe pelo menos um servidor (gerente), denominado ConectaGerente, e podem haver vários clientes (agentes), denominados de ConectaAgente de acordo com a necessidade (Figura 2).

O papel do gerente é gerenciar (comandar) os agentes, através de ações criadas por um operador no gerente. Toda e qualquer ação que o operador criar, será enviada para o agente, que por sua vez deve executá-la e retornar uma resposta da execução.

O agente é gerenciado por um único gerente, que mantém as informações de programas nele instalados.

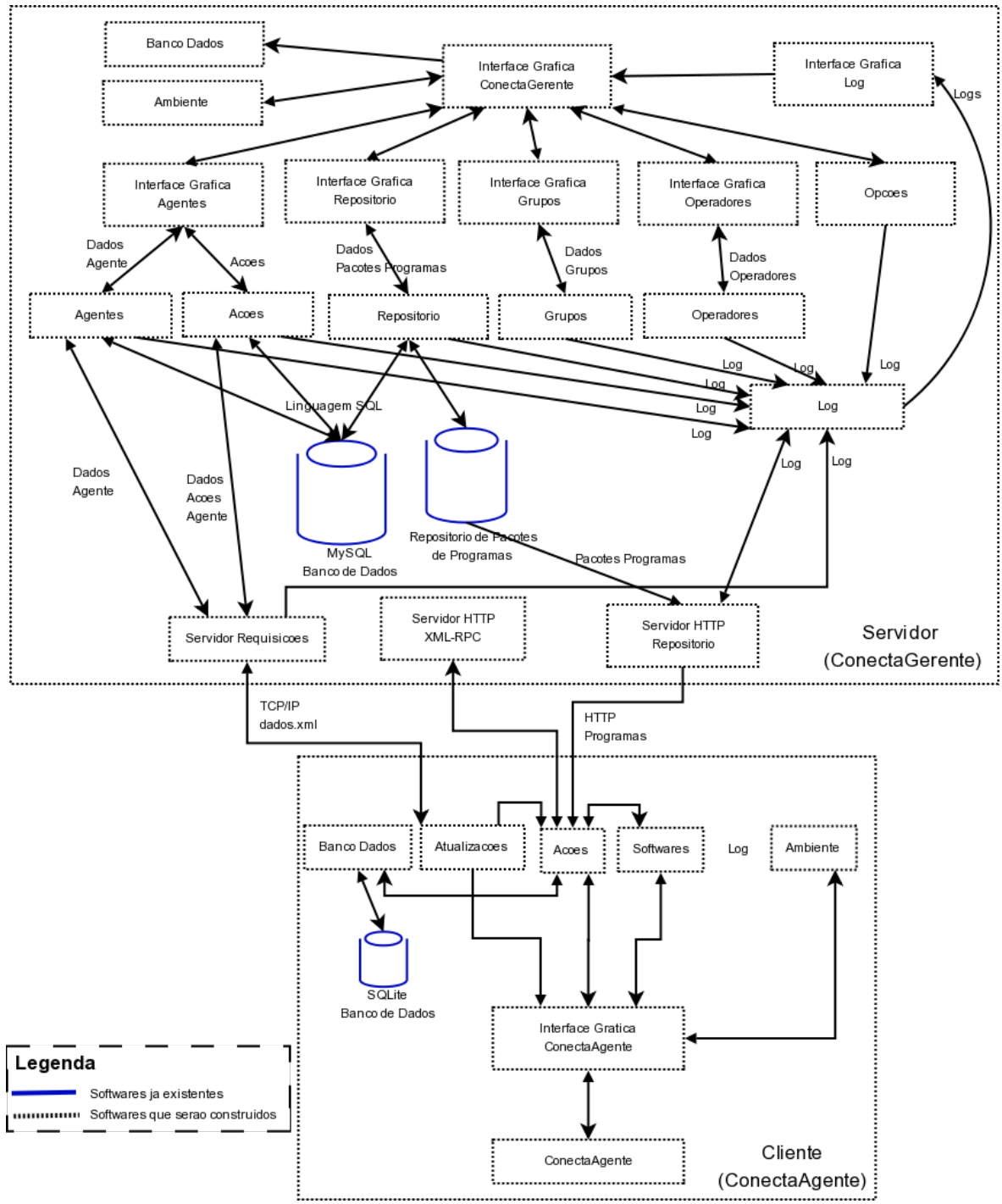

Figura 1: Diagrama de Arquitetura

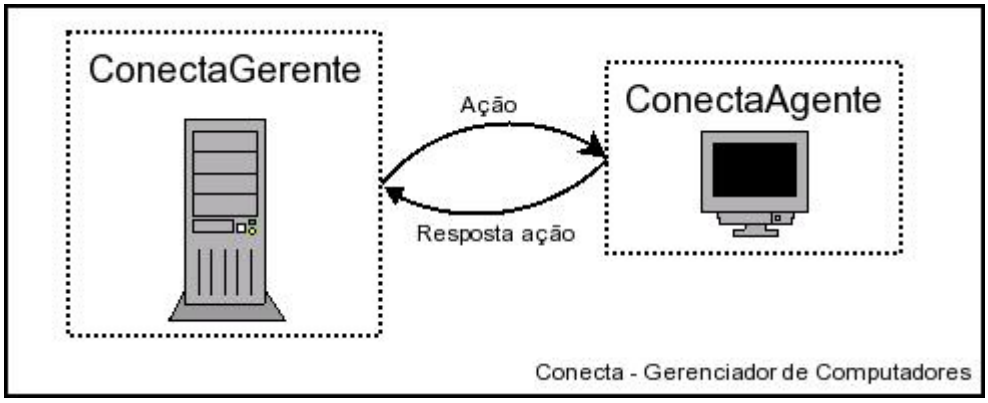

Figura 2: Arquitetura cliente e servidor simplificada

O gerente possui seis partes principais (Figura 3):

- Servidor Requisições, HTTP, RPC,
- Banco de Dados MySQL,
- Gerenciador de Ações, e
- Repositório de Programas.

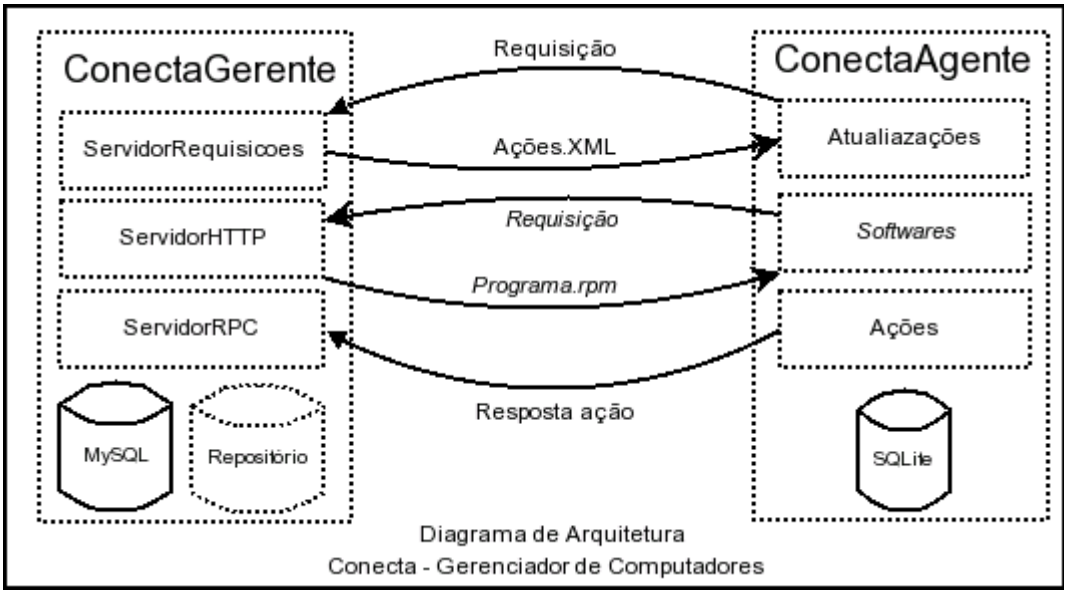

Figura 3: Arquitetura cliente/servidor com os principais módulos

Essa divisão deixa explícitos os módulos do gerente, de maneira a facilitar o desenvolvimento do projeto e sua manutenção.

O gerente e o agente possuem um sistema interno de histórico (*log*), que permite visualizar o que acontece internamente (requisições, ações cadastradas e agentes conectados). Esse sistema grava mensagens em um arquivo texto, sendo que cada mensagem possui a descrição detalhada e a data do acontecimento, possibilitando o administrador detalhar execuções no gerente.

O acesso ao gerente é feito através de uma *interface* gráfica (Figura 4). Essa interface possui uma barra de controle que permite manipular os agentes, o repositório de programas, operadores, grupos (de agentes) e visualizar o histórico de ações executadas.

Através dessa *interface* gráfica, o operador pode cadastrar novos agentes, para que possam ser gerenciados. Esse cadastro possui informações importantes do agente, como o endereço *IP,* o apelido da máquina, o responsável, o endereço físico da placa de rede que o conecta ao gerente (denominado MAC-Address) e escolher ou cadastrar o grupo ao qual o agente vai pertencer.

O grupo do agente deve ser entendido como um perfil de agente, permitindo o operador gerenciar os agentes por perfis, com isso, o operador pode executar uma ação por perfil, aonde a mesma é expandida para todos os agentes pertencentes ao grupo.

Para o agente saber quais ações ele deve executar, ele faz uma busca de ações no gerente através do seu módulo de atualizações, conecta-se diretamente no servidor de requisições do gerente informando o seu *Mac-Address* e o tipo de requisição que ele deseja do gerente.

Caso haja ações disponíveis para serem executadas, o gerente recupera todas as ações do banco de dados cadastradas para aquele agente (de acordo com o *Mac-Address*), gera um arquivo no formato *XML* dessas informações e transfere esse arquivo para agente.

Esse arquivo recebido pelo agente possui as informações necessárias

para que ele entenda do que se trata a ação a ser executada, incluindo a data máxima de execução daquela ação, o tipo de ação, a confirmação de qual agente deve executá-la, o caminho para o agente adquirir o programa (se for o caso) e o operador responsável por aquela ação no gerente.

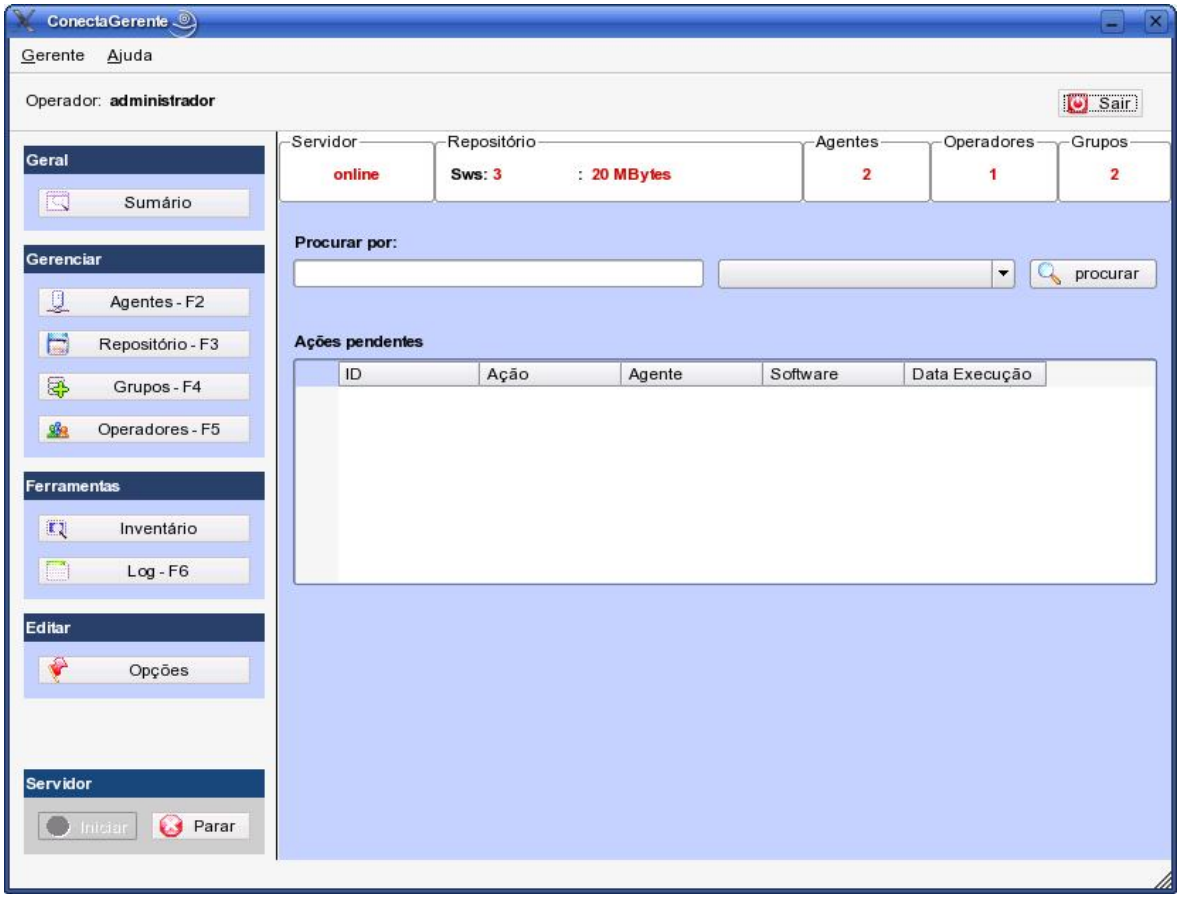

Figura 4: Interface gráfica do ConectaGerente

O agente recebe o arquivo *XML* (Quadro 2) do gerente e recupera todas as informações desse arquivo através de um *parser*. Esse *parser* é uma função interna do agente, o qual permite recuperar informações desse formato de arquivo.

As *tags* desse arquivo *XML* foram definidas especificamente para esse protótipo, levando em consideração as necessidades de troca de informações entre o gerente e o agente.

Com as informações recuperadas, o agente prepara localmente as

ações a serem executadas, inserindo-as no seu próprio banco de dados (SQLite), não necessariamente executando-as instantaneamente, pelo fato do gerente poder configurar um prazo de execução para cada ação.

Quadro 2: Exemplo de arquivo estruturado em XML

```
<?xml version='1.0' encoding='UTF-8'?>
<ConectaGerente xmlns='00:0b:cd:ec:58:84'>
  <instalar>
    <acao_id>5</acao_id>
    <acao_operador>administrador</acao_operador>
    <acao_data_criacao>2006-10-21</acao_data_criacao>
    <acao_data_execucao>2006-11-20</acao_data_execucao>
    <software_nome>libmtp</software_nome>
    <software_versao>0.0.17</software_versao>
    <software_liberacao>1.guru.suse101</software_liberacao>
    <software_tamanho>137611</software_tamanho>
    <software_descricao>Media Transfer Protocol Library</software_descricao>
    <software_grupo>System/Libraries</software_grupo>
    <software_localizacao>repositorio/l/</software_localizacao>
    <software_plataforma>i686-suse-linux</software_plataforma>
    <software_nomeArquivo>libmtp-0.0.17-
1.guru.suse101.i686.rpm</software_nomeArquivo>
    <software_md5>912a91072376ad46d7fd3ff419bf3ba8</software_md5>
  </instalar>
</ConectaGerente>
```

As ações criadas pelo gerente e executas pelo agente podem ser:

- Instalar (programa),
- Desinstalar (programa),
- Atualizar (programa),
- Executar (*script*), e
- Inventariar (programa).

Se dentre as ações adquiridas houver uma nova ação “instalar” (instalação de programa), o agente deve buscar esse programa no gerente. Esse programa é adquirido pelo módulo de ações do agente, através do servidor *HTTP* do gerente. Esse servidor HTTP tem acesso direto ao repositório de programas e os disponibiliza para que os programas possam ser adquiridos e instalados.

Através da interface gráfica do gerente é possível manipular o repositório de programas (Figura 5). Esse repositório é um dos pilares do gerente, pois assim, o gerente consegue catalogar informações dos programas

disponíveis e que são vitais para o sistema funcionar.

Todo o repositório é baseado no sistema RPM. Esse sistema de pacotes permite empacotar programas em um único arquivo compactado e também armazenar nesse arquivo as informações do que há dentro dele (nome, versão, liberação, tamanho, plataforma, descrição e grupo).

Toda vez que um programa é inserido no repositório, o gerente calcula a assinatura do arquivo do programa, para detectar se o programa já está disponível e para o agente verificar se o arquivo que ele solicitou ao servidor é o mesmo que chegou até ele.

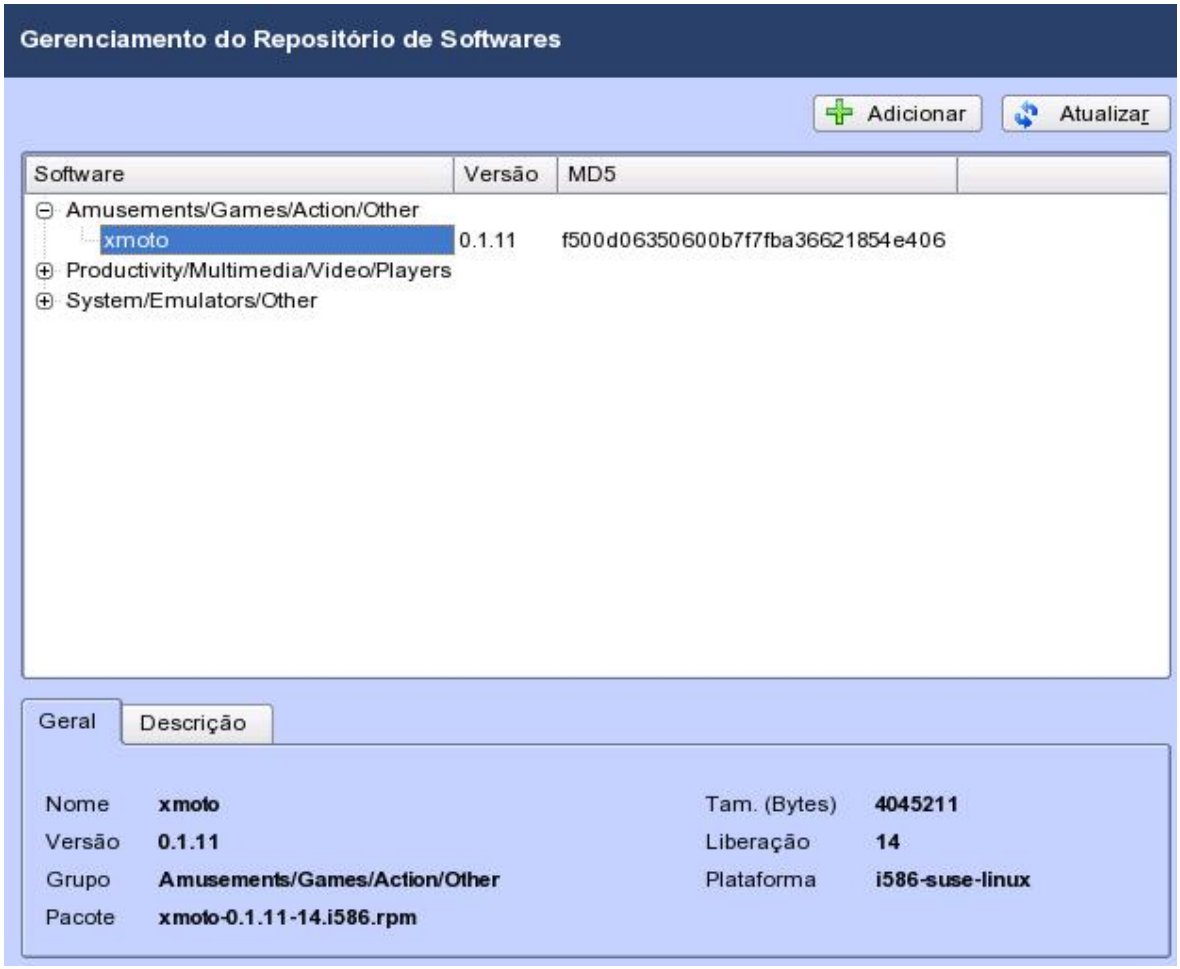

Figura 5: Gerenciamento do repositório de programas no ConectaGerente

O MD5 é um algoritmo calculado através de uma função *hash*, descrita na RFC 1321 (RFC EDITOR, 2006), usado principalmente para a verificação de integridade de arquivo, pois, se qualquer bit do arquivo for alterado ou mesmo trocado de ordem, a assinatura será alterada. O algoritmo que gera este tipo de assinatura é unidirecional, ou seja, não há possibilidade de a partir da assinatura chegar a sequência de bits do arquivo.

Com o nome do programa, versão, MD5 e localização, o agente consegue fazer o *download* do programa dentro do repositório e executar a ação de instalação.

No caso da ação ser “desinstalar”, o agente não precisa buscar o programa no gerente, apenas o remove localmente.

A atualização é o processo sequencial de desinstalar um programa, buscá-lo no servidor HTTP e instalá-lo.

O gerente também pode executar comandos no agente. Esses comandos são arquivos no formato de *script bash*.

A ação de inventariar o agente é a busca de informações de hardware e programas disponíveis nele. Essa ação é extremamente importante, porque possibilita ao gerente saber quais programas já estão instalados no agente e dá o poder para o gerente de monitorar a instalação de programas não disponibilizados pelo gerente.

O inventário é gerado no agente através do módulo *softwares*, que busca as informações de *softwares* e *hardware* do computador cliente. As informações dos *softwares* instalados (nome, versão, tamanho em disco, etc) são obtidas por uma função do módulo rpm, e as informações de *hardware* são obtidas por um *shell script* executado pelo módulo *softwares.* Esse inventário é transferido para o gerente, atualizando os dados cadastrados do agente.

O retorno da execução da ação no agente é através de uma chamada de procedimento remoto XMLRPC (XMLRPC, 2006). Essa chamada é gerada no agente, que, por sua vez, se conecta ao Servidor RPC no gerente, chama uma

função que está no gerente, que recebe como parâmetro a identificação da ação e o endereço Mac Address do gerente.

As atualizações de ações disponíveis para o agente acontecem automaticamente a cada 3 horas, podendo o usuário forçar uma atualização, clicando no botão “Buscar atualizações” na interface do agente.

Através da interface gráfica do agente (Figura 6), é possível visualizar todas as ações (pendentes e executadas), o histórico interno de ações, ao inventário e possibilita o usuário forçar a execução das ações pendentes.

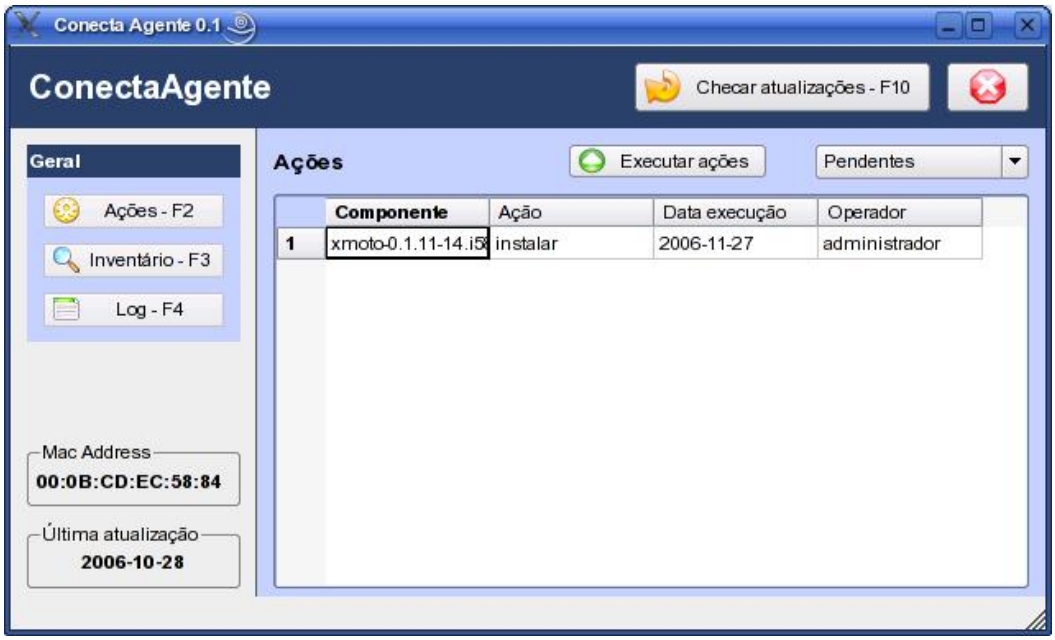

Figura 6: Interface gráfica do ConectaAgente

3.5 Modelo Entidade Relacionamento do Banco de Dados

O modelo entidade relacionamento ajuda no entendimento da estruturação do banco de dados do gerente e do agente.

Através desse modelo, é possível gerar o código da linguagem SQL necessário para criar dentro do banco de dados as estruturas necessárias (tabelas, tipos, relações).

Desenvolver um modelo desse tipo poupa tempo no desenvolvimento do projeto e facilita a manutenção futura.

3.5.1 Gerente

O modelo MER do servidor (Figura 7) mostra a existência de 6 tabelas necessárias para o armazenamento de dados no servidor.

A tabela “agentes“ possui os campos necessários para armazenar os dados de cadastro dos agentes, incluindo os dados de *hardware*, de localização, do responsável, o grupo que o agente faz parte e a identificação única de cada agente (chave primária) “*mac_address*”.

A tabela “grupos” possui os campos necessários para armazenar os dados dos grupos gerenciados no servidor.

A tabela “softwares” possui os campos necessários para armazenar os dados dos *softwares* disponíveis no repositório de *softwares* localizado no servidor. A identificação única de cada *software* é feita pela chave primária *“md5”*.

A tabela “agentes_softwares” é usada para armazenar os dados dos *softwares* instalados nos agentes, a referência do *software* que está instalado em um determinado agente é feita pelos campos “agentes*_mac_address*” (identificação do agente) e “*softwares_md5”* (identificação do *software).*

A tabela “ações” guarda os dados usados no gerenciamento de ações no servidor. Esses dados são criados, consultados e modificados pelo servidor, que também os exporta para um arquivo XML sempre que um determinado agente requisita as ações disponíveis. A identificação única de cada ação é feita pela chave primária “id”.

A tabela “operadores” possui os campos usados para armazenar dados sobre os operadores que controlam o servidor (ConectaGerente). Essa tabela é importante porque possibilita identificar quem ordenou a execução de uma ação.

O gerente é a única entidade que tem acesso a esses dados, já que os agentes não possuem acesso direto, apenas via o Servidor de Requisições do gerente.

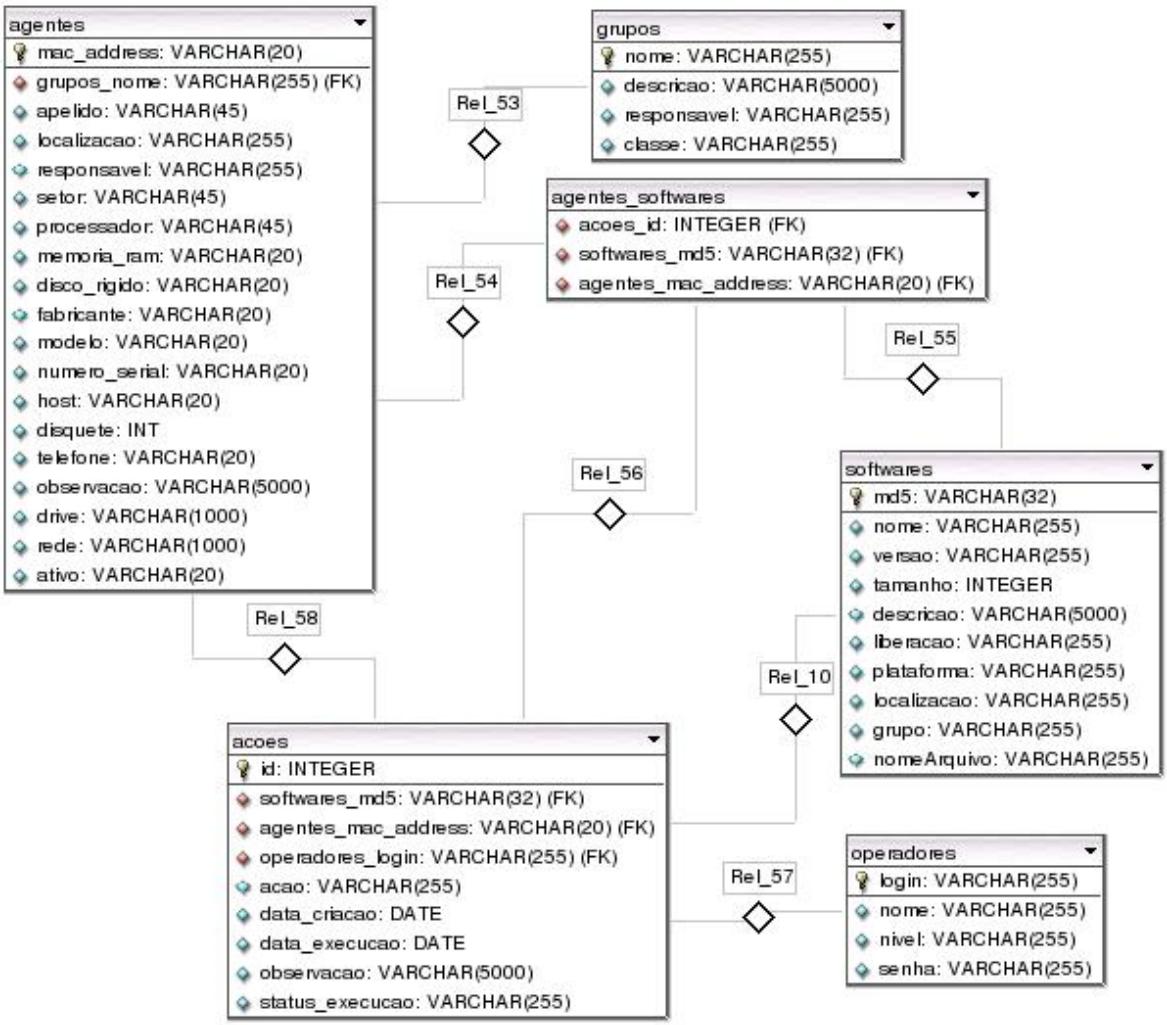

Figura 7: Modelo Entidade Relacionamento - Banco de dados do Gerente

3.5.2 Agente

O agente possui uma estrutura de dados simples (Figura 8), porque a própria arquitetura do projeto não exige que ele guarde informações importantes, apenas informações temporárias.

A tabela “Acoes” do agente possui as informações das ações

executadas e disponíveis para serem executadas pelo agente. A identificação de cada ação é feita pela chave primária “id”, necessária para que o agente possa responder para o gerente se determinada ação foi executada com sucesso.

Os campos da tabela “Acoes” que começam com “*software*_” guardam as informações do *software* necessário para a execução da ação, incluindo o nome do arquivo, o tamanho e o MD5.

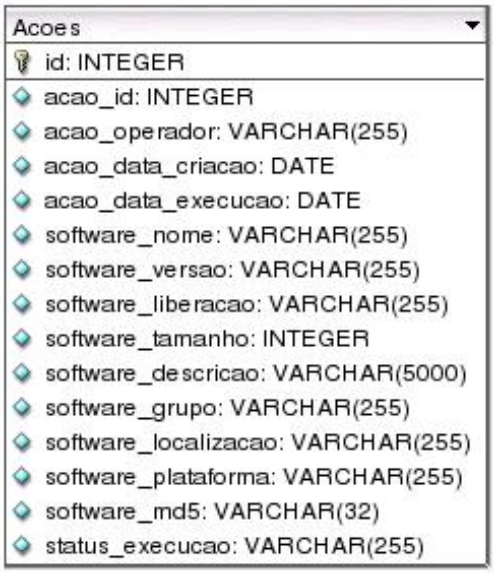

Figura 8: Modelo Entidade Relacionamento - Banco de dados do Agente

4 RESULTADOS

Nesse capítulo é feito o estudo dos resultados do projeto, a avaliação e validação dos dados obtidos com ele, afim de comprovar o objetivo especificado.

4.1 Avaliação

Para avaliar o projeto, três tarefas básicas de manutenção de computadores foram executadas e cronometradas, afim de aferir o tempo gasto por elas.

As tarefas são: instalação, atualização e desinstalação de programas.

Estas tarefas foram avaliadas através de um *shell script* que faz o controle de tempo. Esse *shell script* possui três variáveis: tempoInicial, tempoFinal e tempoTotal. Quando alguma tarefa é iniciada, a variável tempoInicial recebe o horário atual. Após o término da tarefa, a variável tempoFinal recebe novamente o horário atual. Assim a diferença de tempo necessário para a execução da tarefa é dada pela variável tempoTotal em segundos, que recebe o valor de tempoInicial – tempoFinal.

Os dados obtidos para a execução das tarefas levam em conta também o tempo de preparação para a manutenção. Se for manutenção manual, o tempo necessário para adquirir o programa, e se for o caso de manutenção automatizada, o tempo necessário para o cadastro do agente e do programa no gerente. Esse tempo é referenciado como o tempo de preparação.

A avaliação foi feita através da comparação dos tempos para a execução das tarefas manuais e da ferramenta. Para haver uma comparação melhor dos tempos, cada tarefa foi repetida três vezes em agentes diferentes e o tempo total médio foi o tempo considerando para a avaliação.

A avaliação foi feita em um ambiente homogêneo, ou seja, os mesmo processos que estão sendo executados na manutenção manual, estão presentes também na manutenção automatizada pelo projeto, para que não houvesse discrepância na carga do sistema. Para comprovar isso, os resultados estão acompanhados pela carga média do sistema operacional Linux, descritas por três

tempos distintos (último minuto, últimos 5 minutos e últimos 50 minutos). Esses tempos fazem parte do próprio sistema operacional Linux e podem ser obtidos pelo comando “top” (Quadro 3).

O comando top exibe informações do sistema operacional: tempo total que o computador está ligado, carga média do sistema (carga de uso), número de processos (ativos, dormindo e parados), carga do processador, uso de memória ram e *swap*.

Quadro 3: Exemplo da saída do comando TOP

```
top - 19:03:01 up 11:47,  7 users,  load average: 0.09, 0.16, 0.17
Tasks: 119 total,   1 running, 117 sleeping,   0 stopped,   1 zombie
Cpu(s):  7.3% us,  4.6% sy,  0.0% ni, 85.5% id,  2.3% wa,  0.3% hi,  0.0% si
Mem:    499540k total,   486960k used,    12580k free,    39052k buffers
Swap:   297192k total,    48180k used,   249012k free,   195020k cached
```

A avaliação foi feita pelo próprio desenvolvedor da ferramenta. O desenvolvedor está cursando o último período do curso de Engenharia de Computação, é usuário linux desde 1998 e possui experiência na administração de sistemas linux.

Para as tarefas (instalação, atualização, desinstalação), foram selecionados três programas (Figura 9): xMoto, RealPlayer e Snes9x. Esses programas foram obtidos através do *download* via Internet do site da distribuição Linux *Open*Suse (SUSE, 2006).

Software	Versão	MD5
⊖ Amusements/Games/Action/Other		
xmoto	0.1.11	f500d06350600b7f7fba36621854e406
⊖ Productivity/Multimedia/Video/Players		
RealPlayer	10.0.7	3684394f61d816f91fafc6c567854471
⊖ System/Emulators/Other		
snes9x	1.43	a304b3c1e5c5ceed5ea6a833a0a219dc

Figura 9: Programas selecionados para manutenção

O intuito da primeira tarefa é gerenciar manualmente esses programas em três agentes diferentes, selecionando o tempo médio gasto para executar

cada ação, sendo nove instalações, nove atualizações e nove desinstalações de programas, totalizando 27 ações.

A instalação de programas é o processo de aquisição do programa referenciado na ação a ser executada pelo agente, de instalação desse programa e o retorno da execução da ação para o gerente.

A desinstalação de programas é o processo de remoção do programa instalado no agente, referenciado na ação a ser executada.

A atualização de programas é realizada pelo protótipo como uma seqüência de desinstalação e instalação.

4.2 Validação

O projeto obteria êxito se atendesse ao objetivo e diminuir o tempo gasto de manutenção, e se não causasse algum problema que não existia antes.

Os primeiros resultados (Tabela 1) mostram o tempo médio gasto para fazer a instalação de três programas de forma manual em três computadores diferentes.

Tabela 1: Tempo gasto para a instalação de programas manualmente

Programa	Tempo médio (s)			Carga média do sistema (m)		
	Preparação	Execução	Total	-1	-5	-50
xMoto	83	16	99	0.44	0.27	0.27
RealPlayer	78	31	109	0.74	0.89	0.77
Snes9x	86	18	104	0.83	1.10	0.79

Os resultados seguintes (Tabela 2) mostram o tempo médio utilizado para atualizar programas de forma manual os mesmos três programas nos mesmos três computadores.

Finalmente, os tempos médios necessários (Tabela 3) para desinstalar de forma manual os três programas dos três computadores.

Tabela 2: Tempo gasto para a atualização de programas manualmente

Programa	Tempo médio (s)			Carga média do sistema (m)		
	Preparação	Execução	Total	-1	-5	-50
xMoto	92	23	115	0.21	0.57	0.60
RealPlayer	81	45	126	0.69	0.66	0.92
Snes9x	73	27	104	1.05	0.99	0.62

Os próximos resultados estão relacionados com as mesmas ações dos resultados anteriores, mas obtidos através da automatização dessas ações pelo programa protótipo.

Tabela 3: Tempo gasto para a desinstalação de programas manualmente

Programa	Tempo médio (s)			Carga média do sistema (m)		
	Preparação	Execução	Total	-1	-5	-50
xMoto	10	8	18	0.92	1.57	1.14
RealPlayer	08	17	25	0.73	1.06	1.16
Snes9x	11	6	17	0.99	1.10	1.62

Para essas ações, foi necessário primeiramente cadastrar o computador no programa ConectaGerente (tempo de preparação) e instalar o programa ConectaAgente nesse computador, para que o mesmo pudesse gerenciar as ações descritas no gerente.

Posteriormente, foi necessário o cadastro dos três programas no repositório do gerente, para que os mesmos fossem utilizados pelas ações.

Os resultados da instalação automatizada (Tabela 4) são frutos de gerenciamento feito diretamente no ConectaGerente, aonde um operador ordenou a instalação automatizada.

Os tempos médios para a atualização automatizada (Tabela 5) não levam mais em consideração o tempo necessário para cadastrar o computador e o programa no ConectaGerente, visto que isso já foi feito anteriormente.

Os resultados da desinstalação automática (Tabela 6) também não

levam em consideração o tempo necessário para cadastrar o computador e o programa no ConectaGerente.

Tabela 4: Tempo gasto para a instalação de programas pelo protótipo do projeto

Programa	Tempo médio (s)			Carga média do sistema (m)		
	Preparação	Execução	Total	-1	-5	-50
xMoto	45	9	56	0.54	0.47	1.02
RealPlayer	02	15	17	0.68	1.22	0.99
Snes9x	03	10	13	0.71	0.88	0.62

A comparação dos resultados (Tabela 7) demonstra que o protótipo obteve, em todas as ações, um tempo menor para executar a mesma ação de forma manual.

Tabela 5: Tempo gasto para a atualização de programas pelo protótipo do projeto

Programa	Tempo médio (s)			Carga média do sistema (m)		
	Preparação	Execução	Total	-1	-5	-50
xMoto	04	21	25	0.23	0.57	1.31
RealPlayer	03	52	55	0.88	0.66	1.92
Snes9x	02	24	26	0.92	1.20	1.22

A diferença é mais perceptível na instalação de novos programas (Figura 10), aonde o protótipo precisa aproximadamente 75% menos tempo para instalar um mesmo programa instalado de forma manual. Isso pelo fato da instalação manual precisar de mais tempo de preparação para iniciar a instalação.

Tabela 6: Tempo gasto para a desinstalação de programas pelo protótipo do projeto

Programa	Tempo médio (s)			Carga média do sistema (m)		
	Preparação	Execução	Total	-1	-5	-50
xMoto	05	9	14	1.12	0.88	1.13
RealPlayer	03	14	17	1.03	1.22	0.72
Snes9x	04	10	14	1.24	1.30	0.62

Na atualização de programas, a diferença de tempo gasto para executar a ação manualmente e a ação executada pelo protótipo é 70% maior

(Figura 11), justamente por não ser necessária a intervenção manual de busca e localização do programa a ser atualizado, pois este já está cadastrado no repositório do ConectaGerente.

Tabela 7: Comparação do tempo médio gasto para executar ações de forma manual e usando o protótipo proposto pelo projeto

Programa	Tempo médio (s)					
	Instalação		Atualização		Desinstalação	
	Manual	Automatizada	Manual	Automatizada	Manual	Automatizada
xMoto	99	56	115	25	18	14
RealPlayer	109	17	126	55	25	17
Snes9x	104	13	104	26	17	14
Total	312	86	345	106	60	45

Na desinstalação de programas, o tempo necessário (Figura 12) para execução automática é 25% menor, mas não tão discrepante quanto as outras ações, devido ao fato de não ser necessário muito tempo de preparação para executar essa ação.

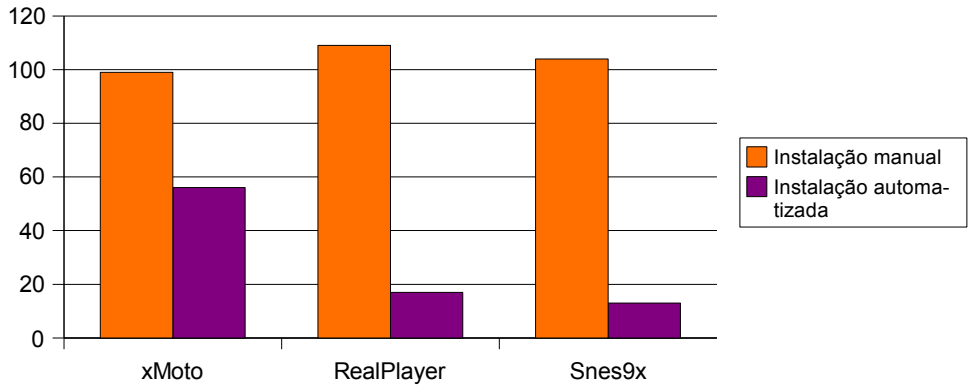

Figura 10: Comparação do tempo gasto (s) para a instalação manual e automatizada

A preparação para executar cada ação corresponde a boa parte do tempo gasto nas ações manuais, aonde é necessário que um técnico ou responsável se desloque até o computador aonde será executada a ação, se autentique no computador (*login*), procure pelo programa a ser gerenciado e inicie

a ação, enquanto que no gerenciamento automatizado, todo esse processo é feito via rede, sem barreiras físicas e de localização.

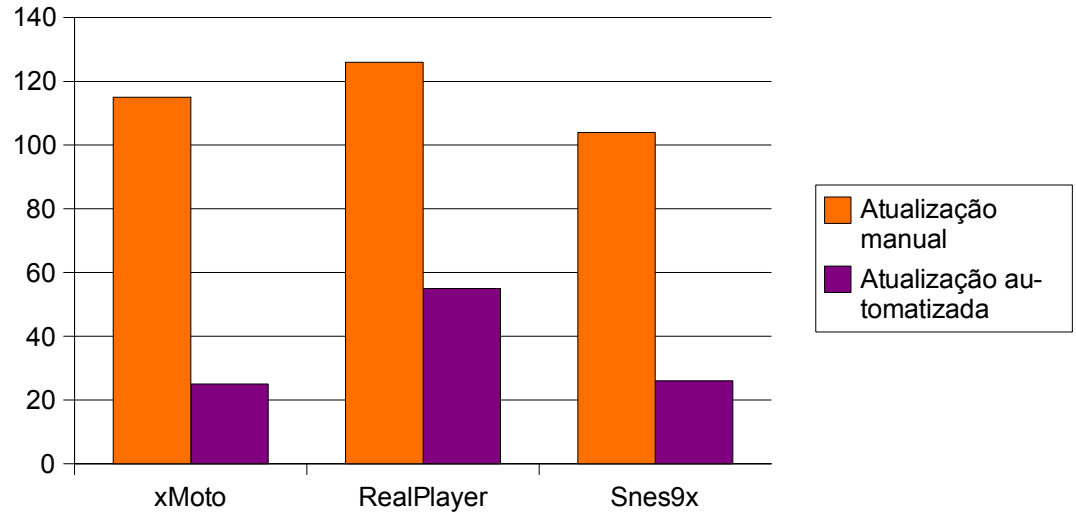

Figura 11: Comparação do tempo gasto (s) para a atualização manual e automatizada

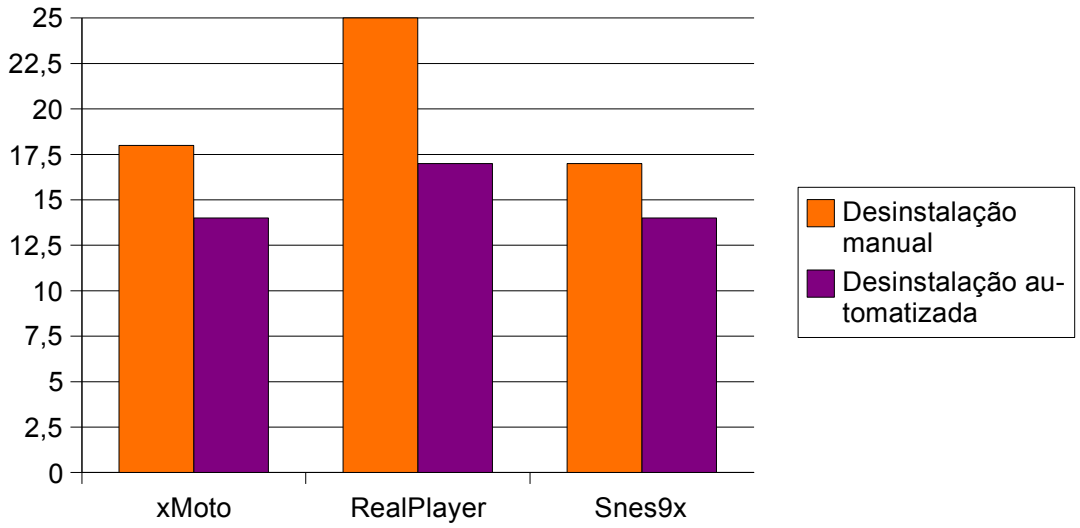

Figura 12: Comparação do tempo gasto (s) para a desinstalação manual e automatizada

O tempo total gasto de manutenção manual para executar as tarefas de instalação, atualização e desinstalação dos três programas nos três agentes foi de 717 segundos contra 237 segundos da manutenção automatizada, isso corresponde aproximadamente 70% de diferença (Figura 13).

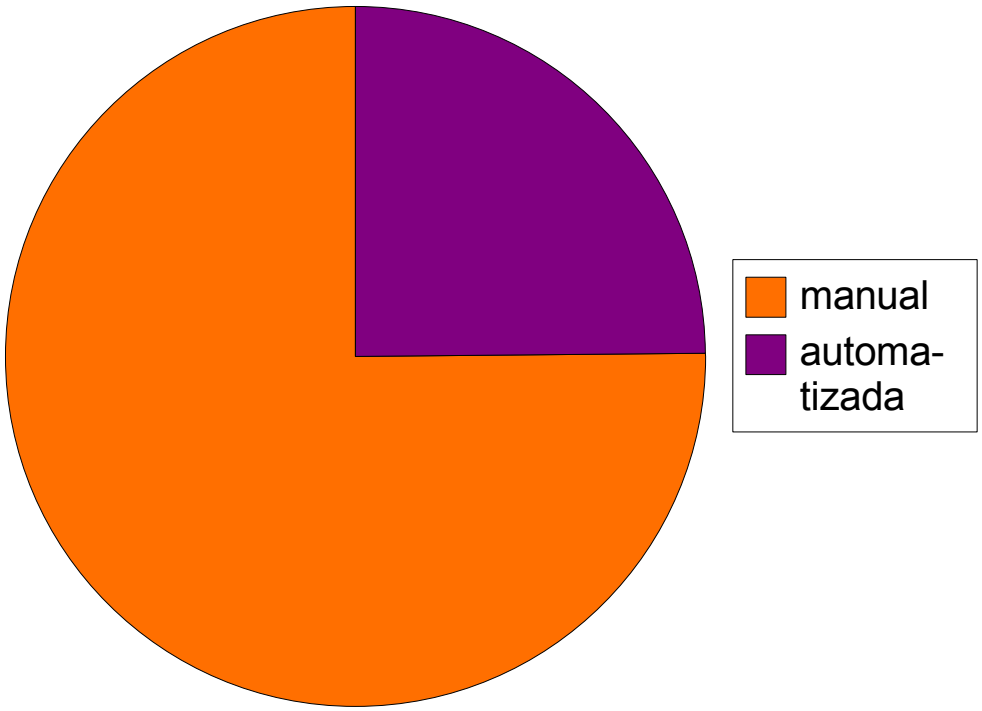

Figura 13: Tempo gasto de manutenção para executar as tarefas de instalação, atualização e desinstalação

5 CONCLUSÃO

Esta monografia foi desenvolvida como parte do trabalho de conclusão do curso de Engenharia de Computação. Este documento teve o intuito de descrever o processo de desenvolvimento de um protótipo que possibilitou resolver um problema atual (manutenção de computadores), as ferramentas necessárias para o desenvolvimento (linguagem de programação, base de dados e bibliotecas) e o resultado obtido com o uso desse protótipo.

O protótipo do projeto Conecta atingiu o objetivo proposto, mostrando ser capaz de automatizar tarefas e diminuir o tempo gasto por tarefas manuais, possibilitando um crescimento escalar de manutenção sem aumentar muito o tempo necessário para fazê-lo.

O tempo necessário para a manutenção pelo projeto chegou a ser 70% menor quando comparado com a mesma manutenção executada de forma manual, comprovando a sua necessidade de existir e também a sua viabilidade.

Disponibilidade e velocidade são dois fatores de lucro, estando o tempo diretamente ligado a esses fatores. Diante disso, automatizar tarefas é uma das maneiras práticas de diminuir o tempo gasto em processos repetitivos.

Os resultados de tempo obtidos na realização manual das tarefas não consideraram que, nessa execução de ações de forma manual, é possível que o responsável técnico, que está executando a ação, erre e/ou não cumpra a ação de forma única e padronizada, ou seja, instale uma versão de programa diferente da que era necessária ou até mesmo remova mais programas do que o necessário.

Por outro lado, o ConectaGerente controla todas as ações de forma automatizada, o que permite ao operador ter ciência do que está sendo feito, contando com que o ConectaAgente não faça nada além do programado pelo operador, padronizando e efetivando todo o processo de manutenção.

O projeto também pode possibilitar a diminuição no número de técnicos responsáveis pela manutenção de vários computadores em uma empresa, visto que parte desse processo é automatizado pelo programa Conecta.

5.1 Dificuldades

No desenvolvimento no projeto foi trabalhoso fazer o controle das ações gerenciadas pelo ConectaGerente, no qual era necessário transformar, formatar e transferir as ações entre os agentes e o gerente, acarretando em grande parte do código desenvolvido.

O pouco tempo de desenvolvimento para o projeto, que devido a sua complexidade, coibiu a adição de novas características e uma boa conciliação com outras matérias do curso.

5.2 Limitações do Programa Protótipo

O protótipo Conecta possui algumas limitações de uso. A primeira é referente ao ConectaAgente, o qual só é possível executar com direito de administrador do sistema, pelo fato do sistema OpenSuse Linux ter controle de permissões diferenciado para administradores e usuários comuns no sistema de arquivos, aonde os programas são gerenciados.

O ConectaAgente também possui uma limitação quanto à resolução de dependências necessárias na instalação de programas, que um programa depende que outro programa já esteja instalado no sistema, o que pode acarretar na não instalação de alguns programas e no retorno de falha pelo próprio agente. Essa limitação é possível de ser resolvida, porém o tempo de implementar essa característica demoraria muito o tempo de desenvolvimento do projeto, impossibilitando o cumprimento do cronograma.

A terceira limitação está relacionada ao número de agentes conectados no ConectaGerente. Atualmente o ConectaGerente não utiliza um controle do número máximo de agentes que podem estabelecer conexões e fazer pedidos. Um número elevado de agentes conectados pode acarretar em uma sobrecarga no gerente, diminuindo o tempo de resposta para todos os agentes nele conectados e até mesmo gerar falhas no sistema.

5.3 Sugestões de Implementação

Algumas características podem ser incorporadas no sistema para melhorar a manuentação de computadores, como a opção de poder instalar o próprio Sistema Operacional através do ConectaGerente, o que resultaria em um controle total dos programas instalados e na redução de tempo.

Uma segunda característica, que agregaria valor ao programa, é a capacidade de lidar com mais de um formato de pacote de programas no repositório do ConectaGerente, possibilitando a migração de sistema operacional ou de distribuição Linux. Pois, atualmente, a ferramenta aceita gerenciar apenas pacotes no formato RPM (desenvolvido apenas para algumas distribuições linux).

Outra sugestão refere-se a melhorar o sistema de controle de autenticação dos agentes, que atualmente é feito pelo endereço da placa de rede (*Mac-Address*), que poderia ser feita usando criptografia assimétrica (chaves privadas e públicas) ou através de códigos de identificação. Se fosse por criptografia, seria possível autenticar que um agente é ele mesmo, através da sua chave privada e da pública.

Outra sugestão é possibilitar ao usuário salvar o perfil de configuração de cada máquina no servidor do Conecta, impedindo a perda de dados e configurações caso o computador cliente (agente) venha a sofrer danos.

6 REFERÊNCIAS

DBDESIGNER. *Banco de Dados*. Disponível na internet em: <http://fabforce.net/dbdesigner4/>. Acesso em: 11 out. 2006.

KDE. *K Desktop Environment*. Disponível na internet em: <http://www.kde.org>. Acesso em: 15 mai. 2006.

NTP. *Network Time Protocol.* Disponível na internet em: <http://www.ntp.org>. Acesso em: 15 mai. 2006.

MYSQL. *Banco de Dados*. Disponível na internet em: <http://www.mysql.org>. Acesso em: 15 mai. 2006.

PYTHON. *Linguagem de programação orientada a objetos*. Disponível na internet em: <http://www.python.org>. Acesso em: 15 mai. 2006.

PYQT. *Python bind for QT*. Disponível na internet em: <http://www.riverbankcomputing.co.uk/pyqt/>. Acesso em: 15 mai. 2006.

RFC EDITOR. *Request for Comments Editor.* Disponível na internet em: <http://www.rfc-editor.org>. Acesso em: 18 dez. 2006.

RPM. *Red Hat Package Manager.* Disponível na internet em: <http://www.rpm.org/>. Acesso em: 13 mai. 2006.

SCITE. *SCIntilla based Text Editor*. Disponível na internet em: <http://www.scintilla.org/SciTE.html>. Acesso em: 18 dez. 2006.

SMART. *Smart Package Manage*r. Disponível na internet em: <http://labix.org/smart>. Acesso em: 07 jun. 2006.

SUSE. *Distribuição Linux*. Disponível na internet em: <http://en.opensuse.org/>. Acesso em: 15 mai. 2006.

SQLITE. *Banco de Dados*. Disponível na internet em: <http://www.sqlite.org>. Acesso em: 15 mai. 2006.

TROLLTECH. *Toolkit Multiplatform GUI & aplication Development* . Disponível na internet em: <http://www.trolltech.com>. Acesso em: 15 mai. 2006.

XMLRPC. *eXtensible Markup Language and Remote Procedure Call*. Disponível na internet em: <http://www.xmlrpc.com/>. Acesso em: 15 mai. 2006.

XML. *eXtensible Markup Language*. Disponível na internet em: <http://www.w3.org/XML/>. Acesso em: 13 mai. 2006.

YOU. *Yast online update*. Disponível na internet em: <http://www.opensuse.org> Acesso em: 15 mai. 2006.