Algumas ferramentas e conceitos utilizados em GED

Nelson Takashi Omori
Orientador: Siang Wun Song

4 de dezembro de 2006

Sumário

Parte I

Monografia

Capítulo 1

Introdução

Este trabalho é baseado em um estágio feito na empresa Murah Technologies. O meu objetivo foi estudar e adicionar funcionalidades no sistema de gerenciamento eletrônico de documentos (GED) que eles desenvolvem.

Como o GED é um sistema que trabalha com várias tecnologias de diversas áreas da computação, pretendo apresentar alguns temas que tive que estudar no decorrer do estágio.

1.1 O que é GED?

GED, Gerenciamento Eletrônico de Documentos, é uma tecnologia utilizada por milhares de empresas espalhadas pelo mundo para capturar, gerenciar, guardar, preservar e disponibilizar documentos relacionados a uma organização.

Algumas vantagens dessa tecnologia são bastante claras, como por exemplo:

- Redução do uso de papel na organização.
- Maior controle sobre os documentos da empresa.
- Economia de espaço físico no armazenamento.
- Permite que mais de uma pessoa tenha acesso simultâneo ao documento.
- O documento pode ser acessado remotamente via web browser.

O GED portanto tem a missão de fornecer à empresa a capacidade de gerenciar todo o capital intelectual da empresa.

No início, a tecnologia enfatizava a digitalização de documentos, que posteriormente poderia utilizar o OCR (Optical Character Recognition) para o reconhecimento de caracteres. Porém, atualmente as empresas geram uma grande quantidade de documentos que já nascem eletrônicos. Com isso surgem vários problemas, como duplicação de arquivos, várias versões de um arquivo, perda de arquivo, etc. e o GED se tornou uma importante ferramenta para gerenciá-los.

Devido aos avanços e popularização da tecnologia, pode-se dizer que hoje o GED é um leque em constante abertura, devido aos diversos serviços possíveis no gerenciamento de documentos de uma organização.

1.1.1 Tecnologias escolhidas para estudo

Dentre as várias tecnologias relacionadas ao GED, foram escolhidos o framework struts, certificação digital e OCR (Optical Character Recognition).

Framework Struts

Hoje em dia, muitas aplicações são baseadas em web. Para facilitar a implementação delas, existem muitos frameworks com o objetivo de facilitar o desenvolvimento do software. Dentre vários existentes, um dos que mais se destaca é o struts, um framework para desenvolvimento web do projeto Apache. Fazendo um estudo mais aprofundado dele, será possível entender por que ele faz tanto sucesso, quais as vantagens utilizando-o, o que é necessário para construir um framework e estudar a estrutura de um projeto bem feito, utilizada por milhares de desenvolvedores.

Certificação Digital

Uma grande preocupação das empresas hoje, é com a validade legal dos documentos gerenciados pelo GED. A certificação digital não supre totalmente esse problema. No entanto, com ela, uma boa parte dos documentos têm validade legal ou pelo menos reconhecidas por outras entidades. Essa tecnologia com certeza será muito utilizada no futuro, e será muito interessante estudá-la com mais profundidade.

Reconhecimento de caracteres

O OCR, como dito anteriormente, faz parte do GED desde o começo de sua utilização. Ela é estudada pela área de visão computacional e utiliza vários conceitos dela.

Framework

Como o GED está relacionado às várias tecnologias, o uso do framework é muito intenso. A utilização do framework Struts é apenas um exemplo disso. O entendimento sobre esse tema é muito importante para o desenvolvimento de um GED e nesse texto especificamente, ajuda a entender o famoso framework do projeto Apache.

Capítulo 2

Framework

Grande parte do custo do desenvolvimento de software se deve à "reinvenção da roda". Desenvolvedores perdem muito tempo descobrindo, inventando e implementando soluções que já é de conhecimento de outros profissionais. Framework é uma técnica de reuso orientado a objetos que tem a função de atenuar essa perda e melhorar a qualidade do software.

2.1 O que é framework?

Podemos definir framework como uma aplicação semi-completa que pode ser especializada por um desenvolvedor. Portanto, frameworks são feitos para atuar em um domínio específico como GUI (Graphical User Interface), MVC(Model-View-Controller), etc.

Em orientação a objetos temos muitas técnicas de reuso de código, como a hierarquia. O framework faz mais do que isso, ele faz o reuso de um projeto comprovadamente que funcional, criada por profissionais especializados na área.

Os principais benefícios da utilização de um framework são:

- **Modularidade** - um usuário de framework trabalha com uma interface estável, que esconde os detalhes da implementação. Essa modularidade melhora a qualidade do software pois reduz o esforço necessário para entender e manter o software.

- **Reusabilidade** - as interfaces estáveis do framework definem componentes genéricos que podem ser reutilizado em várias aplicações.

- **Extensibilidade** - é muito comum a utilização de "métodos ganchos" em um framework. Ele permite que o usuário extenda a interface estável e é essencial para possibilitar a especialização.

- **Inversão de controle** - frameworks detém o controle do fluxo do programa, ao contrário de programas convencionais onde o código principal é do programador. Portanto um usuário de framework apenas conecta o seu código no framework e ele é chamado de acordo com o projeto dos desenvolvedores do framework. Ou seja, ele segue o princípio de Hollywood: Não nos ligue, nós ligamos para você.

Há dois tipos de frameworks: caixa preta e caixa branca. O primeiro, como o próprio nome já sugere, o usuário não necessita ter conhecimentos aprofundados da implementação. Quando o framework é caixa preta o bastante, o programador apenas instancia classes existentes e os conecta. Esse tipo de framework é mais fácil de usar, pois o usuário não perde tanto tempo estudando como utilizá-lo. E quanto mais o framework for caixa-preta, menor será o acoplamento com o código do usuário. No entanto eles são mais difíceis de desenvolver pois os desenvolvedores precisam definir interfaces que devem ser capazes de se encaixar nos diversos casos de uso do framework.

A utilização de um framework caixa branca é muito mais difícil e requer mais tempo de estudo do usuário, pois ele precisa ter conhecimento dos detalhes da implementação. Por outro lado eles são mais fáceis de utilizar e poderosos, quando utilizados por especialistas da área. Em geral, as primeiras versões de um framework são caixa brancas. Com a utilização dele, os desenvolvedores vão obtendo mais conhecimento sobre o problema e o melhora tornando-o mais caixa preta.

2.2 Outras técnicas de reuso e o framework

Framework é apenas uma de muitas técnicas de reuso. Nesta seção comparamos o framework com outras técnicas e analisaremos as diferenças, vantagens, desvantagens e a relação deles com o framework.

2.2.1 Biblioteca

Uma biblioteca de classes é apenas um conjunto de classes que o programador pode utilizar quando necessita. Ao contrário dos frameworks, essas classes não precisam necessariamente estar conectados. Um usuário pode utilizar um dos elementos sem ter nenhum conhecimento dos outros.

Além disso os frameworks têm um campo de atuação muito mais especializado. Por exemplo, classes que são de bibliotecas como String, Array, Coleções, etc. têm uso em aplicações muito diferentes. Porém o poder de reusabilidade dos frameworks é maior.

Outro fator importante é a inversão de controle presente nos frameworks. Bibliotecas são consideradas passivas, isto é, o seu processamento é feito quando o programa principal o chama. Por outro lado, os frameworks são chamados ativos, eles detêm o controle do fluxo do programa.

2.2.2 Componentes

A técnica ideal de reuso é aquela que é facilmente conectada para produzir um novo sistema. O desenvolvedor de software não necessita saber nada da implementação e a especificação do componente é fácil de entender. O sistema resultante será eficiente, de fácil manutenção e estável. No entanto um componente assim é muito difícil de desenvolver. Em geral, quanto maior a simplicidade do uso, menores são os casos que é possível utilizá-lo.

Frameworks são componentes no sentido que eles são vendidos como produto por empresas e uma aplicação pode usar vários frameworks. Porém frameworks são muito mais customizáveis. Como consequência disso o uso do framework dá mais trabalho mesmo quando o desenvolvedor já conhece o framework. Além disso o aprendizado de um novo framework é muito mais difícil. Por outro lado frameworks são muito mais poderosos.

Ao invés de pensar em componentes e frameworks como concorrentes, é melhor considerá-los tecnologias que se cooperam. Primeiro, componentes podem ser desenvolvidos a partir de um framework. Segundo, um framework pode utiliza componentes na sua implementação.

2.2.3 Padrões

Padrões são soluções recorrentes para desenvolvimento de software em um determinado contexto. Padrões e frameworks facilitam o reuso capturando estratégias de desenvolvimento de sucesso. Porém, uma diferença é que frameworks focam no reuso do projeto, algoritmo e implementação em uma determinada linguagem. Por outro lado, padrões focam no reuso de projetos abstratos e micro-arquiteturas de software.

Frameworks podem ser considerados como um subconjunto de padrões que foram codificados para um determinado contexto. Padrões, por sua vez, são mais abstratos, eles não podem ser expressados como uma classe escrita em uma determinada linguagem.

2.3 Considerações finais

Quando utilizadas em conjunto com padrões, componentes e bibliotecas, frameworks podem aumentar significamente a qualidade do software e reduzir o esforço de desenvolvimento. No entanto, há diversos pontos que devemos prestar atenção:

- **Esforço de desenvolvimento** - desenvolver um software complexo é difícil. Produzir um framework de alta qualidade, reusável e extensível é ainda mais difícil. São poucos os desenvolvedores com habilidades necessárias para produzir um framework com sucesso.
- **Curva de aprendizado** - frameworks são mais difíceis de aprender pois o usuário precisa entender todo ele. Dependendo da complexidade do uso, é necessário cursos e dependendo do tempo que levar e do investimento, um framework não seja tão vantajoso assim.
- **Limitações** -

Capítulo 3

Framework Struts

O struts é um framework para aplicações web que segue o famoso padrão Model 2/MVC. Ele é um projeto open source estável, muito bem documentado e é o mais utilizado da categoria na linguagem Java.

Aplicações web são considerados o futuro do software. Algumas vantagens da aplicação web:

- Com ele o usuário final necessita apenas de um browser.
- a atualização e o controle ficam centralizados no servidor, facilitando o controle e a atualização do programa.
- por ser web, ele torna o software "multiplataforma", pois a interface com o usuário é feita por um browser.
- ele pode ficar acessível em qualquer parte do mundo que tenha conexão com a internet.

Porém ele traz alguns problemas:

- O HTTP foi projetado para lidar com conteúdo estático.
- O cliente não guarda estado.
- O servidor não atualiza o cliente.
- O controle de fluxo fica mais difícil.

Ao contrário de sites normais que exibem apenas páginas estáticas, aplicações web devem gerar páginas dinamicamente, de acordo com o usuário, as ações feita por ele e o estado atual da aplicação.

No começo esse serviço era feito por CGIs (Common Gateway Interface), que é um protocolo padrão que faz a interface entre aplicativos e servidor web. Depois, na linguagem Java, surgiu o servlet que era mais rápido e poderoso. Porém a criação de páginas HTML com ele era uma tarefa muito dispendiosa. Para resolver esse problema surgiu o JavaServer Pages (JSP). Com ele desenvolvedores podem misturar códigos Java com HTML e ele virou um sucesso. No entanto isso causou um problema deixando as aplicações web em Java centrado nos JSPs. A solução desse problema não foi criando uma nova tecnologia, mas juntando servlets e JSPs. A responsabilidade do primeiro é o controle do fluxo, enquanto que o segundo cuida apenas da apresentação, ou seja, criação de HTML.

Esse modelo foi denominado Model 2, que tem muitas semelhanças com o padrão MVC. O framework Struts foi criado para fornecer aos desenvolvedores Java uma arquitetura padrão inspirado nesse modelo.

3.1 A base para entender o Struts

Para entender melhor o framework struts, vamos estudar um pouco mais sobre servlet, JavaServer Pages (JSP) e o padrão MVC. Não vamos entrar detalhes nestes temas, mas eles são peças importante para entendermos melhor o Struts.

3.1.1 Model-View-Controller (MVC)

A principal função do padrão Model-View-Controller é isolar o modelo da interface do usuário. Com isso as mudanças, tanto na interface como no modelo, ficam mais fáceis. Principalmente as mudanças na interface, que mudam constantemente em uma aplicação.

Além disso, facilita o desenvolvimento, manutenção e extensão de aplicações que necessitam de várias interfaces para exibir os mesmos dados. Por exemplo um usuário pode acessar o sistema via web ou uma aplicação cliente instalada em sua máquina. Utilizando o padrão MVC, podemos utlizar o mesmo modelo para diferentes interfaces.

O padrão MVC, divide a aplicação em três componentes: modelo, visão e controlador. O primeiro contém os dados e a lógica do negócio. A visão exibe as informações do usuário. O controlador trata as entradas do usuário. A interface com o usuário é formado pela visão e controlador. Um mecanismo de propagação de mudanças, geralmente implementada utilizando o padrão Observer ou Publisher-Subscriber, garante a consistência entre a interface do usuário e o modelo.

Modelo

O modelo contém o núcleo funcional da aplicação. Ele encapsula os dados e o comportamento da aplicação. O controlador acessa as operações disponíveis de acordo com a entrada do usuário. O modelo também disponibiliza uma função para que a visão adquira os dados a serem exibidos.

Ele também disponibiliza um registro dos componentes que dependem do estado do modelo. Quando seu estado muda, ele notifica esses componentes dependentes através do mecanismo de propagação de mudanças. Esse mecanismo é o único vínculo entre o modelo e visão e controlador.

Visão

A visão apresenta a informação para o usuário. Uma aplicação pode ter várias visões e cada um deles precisa implementar um método que é chamado pelo modelo quando tem alguma modificação nele. Esse método adquire as informações necessárias no modelo e o apresenta do seu modo ao usuário.

Cada visão tem um controlador associado a ele. Visões geralmente disponibilizam funções que permitem ao controlador manipular a exibição da informação. Isso é útil para entradas do usuário que não afetam o modelo, como o rolamento da barra de rolagem.

Controlador

O controlador recebe as entradas do usuário como eventos. Eles são traduzidos em requisições para o modelo ou para a visão associada.

Em muitas aplicações o comportamento do controlador depende do estado do modelo. Portanto o modelo deve permitir que o controlador se registre para receber notificações sobre mudanças no modelo e o controlador deve implementar um método para recebê-los.

Dinâmica

A dinâmica da execução do MVC começa com o controlador recebendo a entrada do usuário como um evento e o traduzindo para uma requisição ao modelo. O modelo recebe essa requisição e executa um procedimento, resultando em uma modificação no seu estado. Ele então notifica todos as visões e controladores que se registraram. Cada visão faz a requisição dos dados ao modelo e os exibe ao usuário. Os controladores, ao receberem a notificação, habilitam ou desabilitam funções disponíveis para o usuário.

Conseqüências do uso do MVC

O MVC permite que um modelo tenha várias visões, pois ele separa o modelo da interface com o usuário. Com isso um único modelo pode trabalhar com várias visões ao mesmo tempo.

Essa separação também traz facilidade na manutenção do software. Você pode mudar um dos elementos do padrão e o impacto que isso gera nos outros é muito baixo. Além disso isso permite ao usuário trocar a visão quando quiser, em tempo de execução.

O mecanismo de propagação de mudanças garante que todas as visões e controladores estejam sincronizados com o modelo.

Por outro lado o MVC tem alguns pontos fracos. Um deles é a complexidade de implementá-lo. Dependendo do sistema, a dificuldade dificuldade não compensa os ganhos. Muitas vezes é necessário fazer algumas adaptações no padrão de acordo com o contexto. No entanto esse padrão permite que desenvolva um framework que serve de base para ele.

Outro ponto fraco é o número de notificações que o modelo pode enviar. Quando uma ação do usuário resulta em mudanças seqüenciais no modelo, ele pode enviar muitas notificações para as visões e controladores, sendo que a maioria deles não sejam necessários. Dependendo da interface do modelo, uma visão pode ser obrigado a fazer várias requisições para poder exibir os dados para o usuário. Quando as atualizações são muito freqüentes, haverá muitas requisições desnecessárias de dados que não alteraram.

3.1.2 Servlet

Servlet é um objeto escrito em Java que tem a função de receber uma requisição, construir uma resposta e enviá-lo de volta. Com ele podemos criar páginas da web dinamicamente e interagir com o usuário. Como é escrito em Java, o desenvolvedor pode utilizar todo o poder e flexibilidade dessa linguagem.

O Struts utiliza o servlet para implementar o Controlador do padrão MVC. Como ele é um código Java puro, é muito fácil implementar esse tipo de componente.

3.1.3 JavaServer Pages (JSP)

JavaServer Pages (JSP) é uma tecnologia utilizada para construir aplicações web que lidam com conteúdo dinâmico. Um JSP é formado por elementos HTML e/ou XML (específicos do JSP) e/ou scriplets (pedaços de código Java). Na realidade um JSP é uma extensão de servlet e herda todo o seu

poder. Na primeira requisição para a página JSP, ele é analisado gramaticamente e criado um servlet. Portanto, na visão do usuário, trabalhar com JSP ou servlet não tem diferença.

Como o JSP permite misturar elementos HTML, ele se torna uma ferramenta muito fácil para construir páginas web dinâmicas. No Struts ele é geralmente utilizado para implementar a Visão do padrão MVC.

3.2 Struts

Agora que temos uma noção dos temas básicos, podemos realmente entrar no estudo do struts.

3.2.1 Model 2/MVC

Vimos anteriormente como o MVC, seus componentes e o funcionamento. No entanto algumas coisas não são aplicáveis quando se trata de aplicação web.

Uma delas é a notificação do modelo para visões e controladores quando seu estado muda. Em uma aplicação web o servidor não pode atualizar o cliente, fazendo com que a visão não fique sincronizado com o modelo.

Portanto, o padrão utilizado no Struts não pode ser considerado um MVC "puro", sendo denominado Model 2/MVC.

3.2.2 Estrutura

Nesta seção vamos estudar algumas classes importantes do framework Struts e como eles se relacionam.

Controle de fluxo com o componente ActionServlet

No framework Struts existe apenas um componente controlador, que é a classe ActionServlet. Ele representa o Controlador do padrão MVC e também os padrões Front Controller e Singleton. O primeiro possibilita um ponto de acesso centralizado para as requisições da apresentação. O segundo padrão garante que apenas uma instância do objeto vai ser criado por aplicação. Esse único objeto recebe e processa todas as requisições que muda o estado da interação com o usuário.

O ActionServlet seleciona e chama a classe Action apropriada, que é implementada pelo usuário do framework, para executar a requisição para a camada da lógica de negócio. As classes Action não produzem diretamente a

próxima página da interface com o usuário. Eles utilizam o RequestDispatcher.forward() do servlet API para passar o controle para o JSP apropriado, que por sua vez produz a página para o usuário.

A escolha da classe Action correta é feita analisando o endereço chamado pelo usuário. Esse mapeamento é feito usando um arquivo de configuração em XML chamado struts-config.xml. Depois de identificado a classe Action correta, o ActionServlet verifica se já existe uma instância dele. Se existir ele usa essa instância, caso contrário instancia um objeto dessa classe e vai ser guardado para uso no futuro. Observe que desse modo vai existir apenas uma instância de cada classe Action. Portanto o desenvolvedor precisa desenvolvê-lo de modo que seja thread safe. Construir uma classe que utiliza apenas variáveis locais já garante isso.

Se na configuração é especificado um objeto do tipo ActionFormBean, que serve para transportar os dados inseridos pelo usuário, ele é instanciado e preenchido com valores padrões. Opcionalmente vai ser chamado o método validate(), que como o próprio nome já diz, valida os valores dos campos de ActionFormBean.

O método principal de um Action é o execute(). Quando um usuário implementa um Action, ele deve implementar esse método que chama o elemento apropriado da camada de negócios e indica qual página deve ser criada.

Trabalhando com os componentes do modelo

O M do padrão MVC contém os componentes do modelo. Os componentes do modelo são aqueles que representam o estado da aplicação. Nessa camada é mantida a lógica de negócio específica da aplicação.

ActionMapping O ActionServlet necessita de um mecanismo para determinar como escolher os Actions corretos para cada requisição e a classe ActionMapping representa a informação que o ActionServlet sabe do mapeamento de um Action em particular. Esse mapeamento é passado como parâmetro no método execute(), assim o Action pode acessar as informações diretamente. Segue um exemplo de como o mapeamento é feito no arquivo struts-config.xml:

```
<action-mappings>
 <action path=''/formularioUsuario''
         type=''br.usp.ime.linux.FormularioUsuarioAction''
 name=''formularioUsuarioForm''
 scope=''request''
 input=''/formularioUsuario.jsp''>
```

```
 <forward name=''sucesso'' path=''telaSucesso.jsp'' />
 <forward name=''erro'' path=''telaDeErro.jsp'' />
 </action>
</action-mappings>
```

- O *path* indica parte do endereço associado a esse mapeamento.
- O *type* indica a classe associada a esse mapeamento
- O *name* indica o nome do ActionFormBean associado.
- O *scope* indica o escopo do ActionFormBean.
- O *input* indica a página que deve ser retornada caso seja encontrado algum erro na validação.
- Os *forward*s indicam os nomes internos que um Action pode retornar para controlar qual página exibir ao usuário.

Action Um action é um adaptador entre requisições HTTP e a lógica de negócios que deve ser executada. Podemos pensar um Action como se fosse uma cola entre a requisição do cliente e a lógica do negócio que deve ser executada.

ActionForward Representa um destino que um ActionServlet pode ser direcionado. O método execute() de um Action deve devolver um objeto do tipo ActionForward indicando que página deve ser exibido. Ele pode ser instanciado dinamicamente ou configurado em associação com um ActionMapping.

ActionErrors O mecanismo para retornar erros que ocorre durante a validação de um ActionForm utiliza a classe ActionErrors. Ele encapsula um erro retornado pelo método validate() de um ActionForm. O erro consiste em uma chave que serve para pegar a mensagem em um arquivo resource. Com esse esquema, podemos programar toda a aplicação independente do idioma. Caso seja necessário adicionar um idioma, basta criar mais um arquivo resource que contém as chaves e o texto no idioma.

ActionMessage Ele funciona examatente da mesma forma que o ActionErrors, com a diferença que ele pode ser usado para passar mensagens que não são erros.

Componentes da Visão

Os componentes da visão consiste nos JSPs criados para a aplicação, os ActionForms que podem existir ou não a cada página e o custom tag fornecido pelo Struts.

ActionForm Para cada JSP que tem uma entrada de dados do usuário, podemos criar um ActionForm associado a ele. Um ActionForm é um JavaBean que contém campos, e cada um deles tem métodos de acesso get e set. Por exemplo, um ActionForm que tem como campo uma String nome, teria métodos de acesso getNome e setNome(nome).

Depois desses campos serem preenchidos, queremos verificar a validade deles antes de enviá-lo ao Action. Para isso basta sobrescrever o método validate() que retorna um ActionErrors.

Capítulo 4

Certificação Digital

Com o avanço e popularização da tecnologia o modo como as organizações trabalham e realizam negócios estão mudando. Muitos processos são criados e finalizados inteiramente no meio digital, muitas vezes utilizando a internet. No entanto isso traz várias preocupações como a integridade, o sigilo, a autenticidade do documento, entre outros.

A certificação digital é um conjunto de técnicas e processos que propiciam mais segurança às comunicações e transações eletrônicas, permitindo também a guarda segura do documento [8]. Ele é um arquivo de computador que identifica uma pessoa ou entidade normalmente contém uma chave pública, nome e endereço, a validade, a Autoridade Certificadora (CA) que emitiu o certificado, o número de série e a assinatura digital da CA. Com um certificado digital é possível assinar digitalmente documentos, comprovar identidade e criptografar mensagens.

Apesar dele ser baseado em conceitos matemáticos sofisticados, o seu uso é muito simples. A maioria dos software de correio eletrônico e navegadores já estão preparados para trabalhar com certificados.

4.1 Teoria da certificação digital

A base da certificação digital é a criptografia de chave pública. Porém, antes iremos fazer uma introdução à criptografia. Como dito anteriormente, a certificação digital é um conjunto de técnicas e procedimentos, e iremos ver como é a estrutura que garante toda a segurança e confiabilidade do sistema.

4.1.1 Introdução à criptografia

Algoritmos criptográficos basicamente objetivam "esconder" informações sigilosas de qualquer pessoa desautorizada a lê-las, isto é, de qualquer pessoa que não conheça a chamada chave secreta de criptografia [10].

Estudo com um exemplo

Como motivação e para ilustrar alguns pontos importantes da criptografia, vamos estudar uma das criptografias mais famosas e imediatas que existe: substituição simples. Ele consiste em substituir as letras do texto original por alguma outra letra qualquer. Por exemplo, a seguinte tabela ilustra como é essa criptografia:

texto original: a b c d e f g h i j k l m n o p q r s t u v w x y z
texto cifrado: z p b y j r g k f l x q n w v d h m s u t o i a e c

O algoritmo é simples, basta substituir as letras do texto original segundo a tabela acima. Portanto a palavra FORMATURA seria criptografada como RVMNZUTMZ.

Nessa criptografia, a chave secreta é a tabela. Portanto ela tem $26! \approx 2^{88}$ possíveis chaves. Se considerarmos que alguém intercepte a mensagem criptografada e tente quebrá-la testando todas as possibilidades, mesmo que a pessoa tenha uma máquina que teste 2^{40} chaves por segundo, ela demoraria mais de 8900 anos para quebrá-lo. Portanto, podemos afirmar que a criptografia de substituição simples é segura? A resposta é não, e veremos o porquê logo abaixo.

Criptanálise

Criptanálise é o conjunto de métodos matemáticos para quebrar um algoritmo ou um texto cifrado. Por exemplo, se cifrarmos ABACATE com o algoritmo de substituição simples apresentado anteriormente, obtemos ZPZBZUJ. Observe que as três repetições da letra 'A' também foi transferido para o texto cifrado, representado pela letra 'Z'. Uma pessoa que intercepte essa mensagem e desconfie que ela foi cifrada com o algoritmo de substituição simples, pode fazer a suposição que 'Z' seja uma vogal, 'P' e 'B' consoantes. Considerando isso, as possibilidades diminuem drasticamente e o ataque pode ser feito mais facilmente.

Esse tipo de ataque, utilizando apenas informações do texto cifrado, é chamado ataque por só-texto-ilegível. Um exemplo clássico é o ataque estatístico. Se considerarmos uma língua em particular, podemos fazer estatística de quais letras aparecem com mais freqüência e utilizar essa informação para

decifrar a mensagem ou pelo menos para facilitar. Nesse ataque, quanto mais texto cifrado o atacante tiver, mais próximo ele chegará à quebra.

O exemplo acima mostra que mesmo tendo um número de possibilidades grande, não é o suficiente para determinar que o algoritmo é seguro ou não.

Definição de segurança

Para definir se um algoritmo é seguro, o ideal seria ter uma prova matemática disso. No entanto existe apenas um algoritmo com essa propriedade e ela é impraticável para a maioria dos casos.

Migrando a nossa definição para um cenário mais realístico, definimos como algoritmo seguro aquele onde o melhor ataque conhecido requer o mesmo esforço que testando todas as possibilidades. Portanto devemos selecionar o algoritmo que é seguro (segundo a definição acima) e tenha o número de chaves grande o bastante de modo que o teste de todas as possibilidades seja impraticável. Os dois fatores são necessários.

No entanto a definição acima não parece boa, e o motivo é a expressão "o melhor ataque conhecido". Isso significa que a qualquer momento a criptografia pode ser quebrada? Sim, se alguém descobrir um ataque que requer menos esforço que testar todas as possibilidades.

É por isso que todos os algoritmos criptográficos utilizados hoje são de conhecimento público, para que especialistas do mundo inteiro possam analisar e procurar "buracos" no algoritmo. Só depois de passar por vários processos, a comunidade afirma que o algoritmo é seguro.

4.1.2 Criptografia de chave pública

O exemplo visto anteriormente utiliza chave simétrica, ou seja, a chave usada para criptografar é a mesma usada para descriptografar. Porém, como dito anteriormente, a base da certificação digital é a criptografia de chave pública.

Nessa criptografia são utilizadas duas chaves diferentes, uma para criptografar e outra para descriptografar. Isso possibilita tornar uma das chaves públicas. Quando alguém quer enviar uma mensagem criptografada para outra, basta pegar a chave pública e criptografar. Somente quem tem a chave privada consegue descriptografá-la.

Esse esquema resolve um dos maiores problemas da criptografia de chave simétrica, que é a distribuição segura das chaves entre os participantes. Ele também possibilita criar uma espécie de assinatura, pois os dados criptografados com a chave privada somente podem ser descriptografadas pela chave pública. Se você tem garantias de que a chave pública pertence a uma certa pessoa, você tem certeza de que quem criptografou os dados foi ela.

A criptografia de chave pública é baseada em funções fáceis de computar em uma direção de difícil em outra. A razão disso é garantir que um atacante não consiga utilizar a informação pública para reconstruir a mensagem original. Por exemplo, é relativamente fácil gerar dois primos p e q, e calcular o produto deles. No entanto é computacionalmente difícil encontrar os fatores p e q, dado um número N.

Nessa pequena introdução sobre criptografia de chave pública, concluímos que com ela podemos fazer as mesmas coisas que com a criptografia simétrica e mais outras coisas. Por que então não largamos a criptografia simétrica e usamos somente a de chave pública para tudo? Uma das razões é performance. A criptografia simétrica é mais rápida, fazendo com que atualmente ela seja usada para criptografar a maior parte dos dados atualmente.

Como dito anteriormente, um dos pontos críticos da criptografia simétrica é a distribuição das chaves entre os participantes. A solução para esse problema foi misturar criptografia simétrica com criptografia de chave pública. A chave simétrica é distribuída utilizando a criptografia de chave pública. Feito isso os participantes podem trocar mensagens criptografadas utilizando a criptografia simétrica.

Como citado anteriormente, com a criptografia de chave pública você pode assinar digitalmente os dados, uma vez que os dados criptografados com a chave privada podem ser descriptografadas somente com a chave pública. Portanto qualquer um pode pegar a chave pública, aplicar o algoritmo para descriptografar e verificar o autor da mensagem. Isso é uma das vantagens em relação à assinatura manuscrita tradicional, onde é necessário um especialista para verificar a autenticidade e demora muito tempo para essa verificação. Com a assinatura digital a verificação é instantânea e não tem a necessidade de um especialista.

4.1.3 Não-repúdio

Com a criptografia de chave simétrica, temos a garantia de confidenciabilidade e integridade (utilizando uma técnica chamada MAC, Message Authentication Code). A primeira significa que uma pessoa de fora não terá acesso às informações que estão sendo transmitidas. A outra significa que se alguém interceptar a mensagem criptografada e ao invés de tentar recuperar a mensagem original, tentar alterar o conteúdo da mensagem, o receptor irá descobrir isso. Essas duas garantias a criptografia de chave pública também provê. Mas uma coisa que ela garante e que a criptografia de chave pública não, é o não-repúdio.

Para entender melhor isso, vamos ver um exemplo ilustrativo. Considere dois personagens: Alice e Bob. Suponha que Alice queira investir em ações

e faça uma requisição a Bob para a compra das ações de uma empresa X, utilizando a criptografia simétrica. Logo após Bob ter comprado as ações, o valor delas caem drasticamente. Para tentar se safar do prejuízo, Alice afirma que nunca fez nenhuma requisição a Bob. Ele tem como provar que ela fez as requisições? Não, pois a chave usada para a transação era compartilhada, ou seja, Bob também tinha acesso à ela. Agora, considere o mesmo cenário mas agora eles utilizam a assinatura digital. Alice manda a solicitação de compra assinando digitalmente a mensagem. Se Alice afirmar que não fez tal solicitação, Bob tem como provar que fez, basta apresentar a mensagem assinada. Portanto Alice não pode negar que enviou a mensagem.

4.1.4 Confidenciabilidade e Não-repúdio

Agora considere um novo personagem, chamado Charlie. Imagine que Alice deseja mandar uma mensagem para Bob e deseja confidenciabilidade e não-repúdio. Portanto ela tem duas opções:

- Assinar a mensagem com a sua chave privada e depois criptografá-la.
- Criptograr a mensagem com a chave pública de Bob a assiná-la com a sua chave privada.

Vamos analisar a primeira alternativa. Alice pega a mensagem "Eu te amo", assina com a sua chave privada e a criptografa com a chave pública de Bob. Suponha que Bob descriptografe com sua chave privada e leia a mensagem. Porém Bob não gosta de Alice e pretende colocá-la em uma situação ruim. Ele pega a mensagem assinada por Alice e o criptografa com a chave pública de Charlie e manda para ele. Charlie então recebe a mensagem e acha que Alice o ama. Vendo essa história, concluímos que assinar primeiro e criptografar depois não é uma boa idéia. Temos então a segunda alternativa.

Considere então uma situação onde Alice deseja enviar um projeto elaborado por ela a Bob. Como é um projeto sigiloso e ela deseja receber os créditos sobre o trabalho, ela deseja confidenciabilidade e não-repúdio. Assim ela pega os dados do projeto e criptografa com a chave pública de Bob e o assina com a sua chave privada. Porém, antes da mensagem chegar a Bob, Charlie a intercepta e sabe do que se trata a mensagem. Ele então resolve receber os créditos do trabalho, tirando a assinatura da mensagem usando a chave pública de Alice e assina com a sua chave privada. A mensagem chega a Bob e ele dá os créditos do trabalho ao Charlie.

Concluímos então que nenhuma das alternativas é aplicável. O ponto desse problema é que qualquer um pode executar as operações de chave pública. Veremos na próxima seção como resolvê-lo.

4.1.5 Infra-Estrutura de Chaves Públicas

A ICP (Infra-Estrutura de Chaves Públicas) é a soma de tudo o que é necessário para utilizar de modo seguro a criptografia de chaves públicas.

Um dos elementos é o certificado digital ou certificado de chave pública, que é um arquivo digital que contém o nome da pessoa ou entidade e a chave pública dela. Quem garante a autenticidade do certificado, ou seja, confirma que a chave pública realmente pertence àquela pessoa é uma autoridade certificadora (AC). Para isso ela assina digitalmente o certificado, assim qualquer um pode verificar a autenticidade do certificado. A questão é se o receptor aceita a autoridade certificadora como íntegra. Na prática qualquer um pode assinar digitalmente um certificado, afirmando que as informações contidas nela são verdadeiras. Por exemplo, Charlie pode criar um certificado digital tentando se passar por Alice e ele mesmo assiná-lo. No entanto, Bob pode verificar quem assinou aquele certificado e decidir se confia ou não naquela pessoa.

É fácil notar que existe um problema nesse esquema. Quem garante que a assinatura do certificado é realmente da autoridade certificadora ou se é alguém tentando se passar por ela. Uma solução é o receptor guardar todos os certificados das ACs que ele confia. Porém isso se torna inviável devido à dimensão dessa Infra-Estrutura. A criação e o gerenciamento dos certificados não pode ficar concentrado em um grupo pequeno de autoridades. Por outro lado, deve-se ter um controle sobre as ACs para evitar fraudes. A solução é construir uma cadeia de certificação de forma hierarquizada, ou seja, uma AC certifica outra AC, que certifica outra e assim em diante até que uma certifica uma pessoa. Assim o receptor pode ter um pequeno conjunto de certificados das ACs e mesmo assim verificar se o certificado é válido.

Um ponto importante é quando uma AC comete um engano, como emitir um certificado de Alice para Charlie, ou quando um usuário percebe que teve o seu certificado fraudado ou qualquer outra situação que torne o certificado inválido. No esquema descrito até agora, isso não pode ser verificado. Por exemplo, uma pessoa, de alguma forma, consegue ter acesso à chave privada de uma AC. Assim, ela pode emitir certificados que ela quiser e todas as pessoas que acreditam naquela autoridade passam a aceitar esses certificados.

Para evitar essa situação, cada AC tem uma lista de certificados revogados (LCR) e a deixa pública para que qualquer um possa consultá-la. Portanto, uma pessoa que pretende verificar a assinatura de um certificado, além de verificar toda a cadeia, ela deve procurar nas LCRs para ver se um dos certificados foi revogado.

4.1.6 Funções Hash

As funções hash são muito úteis na criptografia, e uma das aplicações é no cálculo da assinatura digital. Suponha que Alice deseje mandar uma mensagem M e sua assinatura S, que é calculada a partir de M, para Bob. Assim Bob pode, a partir de S, obter M e verificar se os dois são iguais. Porém, se M for muito grande o cálculo de S é muito custoso, sem contar o desperdício de banda utilizada para enviar as duas informações.

Aplicando uma função hash em M, obtemos uma "impressão digital" da mensagem, ou seja, uma informação relativamente pequena mas que identifica M. Assim Alice pode obter S' a partir da "impressão digital" de M, e enviar M e S' para Bob, com a vantagem de que S' é muito menor que S. Bob, por sua vez, pode aplicar a mesma função hash em M e verificar a igualdade do resultado com a informação obtida através de S'.

Uma função hash criptográfica precisa prover ter as seguintes características:

- **Compressão:** Para qualquer tamanho da entrada a saída é pequena. Na prática as funções hash criptográficas produzem uma saída de tamanho fixo.
- **Eficiência:** O cálculo da função deve ser fácil computacionalmente. O esforço necessário para o cálculo da função vai certamente aumentar com os dados muito grande, mas ele não pode aumentar tão rápido.
- **Uma direção:** Dado um valor y, é computacionalmente difícil calcular x tal que h(x) = y sendo h a função hash, ou seja, é difícil de inverter a função hash.
- **Fraca resistência à colisão:** Dado x e h(x), é computacionalmente difícil encontrar y tal que h(y) = h(x).
- **Forte resistência à colisão:** É computacionalmente difícil encontrar x e y tal que h(x) = h(y).

4.2 Certificação digital no Brasil

No Brasil o governo federal criou sua própria infra-estrutura de chaves públicas, o ICP-Brasil. Nele o certificado raiz é o ICP-Brasil raiz, que pode ser obtido no site http://www.icpbrasil.gov.br. O responsável pela administração dela é o Instituto Nacional de Tecnologia da Informação, vinculado à Casa Civil da Presidência da República. Ele tem por competências emitir, expedir, distribuir, revogar e gerenciar os certificados das Autoridades

Certificadoras - AC de nível imediatamente subseqüente ao seu; gerenciar a lista de certificados emitidos, revogados e vencidos; executar atividades de fiscalização e auditoria das AC, das Autoridades de Registro - AR e dos prestadores de serviço habilitados na ICP-Brasil [9].

Compete ainda ao ITI estimular e articular projetos de pesquisa científica e de desenvolvimento tecnológico voltados à ampliação da cidadania digital. Neste vetor, o ITI tem como sua principal linha de ação a popularização da certificação digital e a inclusão digital, atuando sobre questões como sistemas criptográficos, software livre, hardware compatíveis com padrões abertos e universais, convergência digital de mídias, entre outras [9].

Capítulo 5

Reconhecimento de caracteres

Reconhecimento de caracteres é o processo de conversão de imagem em texto editável por um computador. Nas últimas décadas a quantidade de documentos gerados em papel foi, e ainda é, muito grande. Porém, hoje devemos tornar os processos ágeis fazendo com que os computadores possam ter acesso e gerenciar estes documentos. O primeiro passo é digitalizá-los utilizando um scanner. Feito isso, temos apenas uma imagem como outra qualquer. Para extrair o texto do documento, passamos o software de reconhecimento de caracteres.

Os primeiros softwares de reconhecimento foram os OCRs (Optical Character Recognition) que reconhece letras geradas por máquinas, ou seja, em uma determinada fonte. Atualmente esses softwares têm um índice de acerto perto de 90%, quando a imagem é de boa qualidade.

O próximo passo foram os ICRs (Intelligent Character Recognition), que lidam com um problema bem mais complexo: letras manuscritas. Esta complexidade é devida as variações de escrita pessoa para pessoa. Até mesmo a mesma pessoa pode escrever a mesma letra de formas diferentes. A escrita manuscrita pode ser dividida em duas formas: isolada ou cursiva. Na primeira, as letras se apresentam de uma forma não conectada. Na segunda as letras podem estar conectadas ou não.

Neste texto vamos nos concentrar mais nos OCRs, falar um pouco da sua história e um pouco da teoria, sem entrar em muitos detalhes.

5.1 A história do OCR

Na década de 50 a revolução tecnológica começou a se mover tão rápido que o processamento de dados eletrônicos se tornou um campo muito importante. A entrada de dados era feita através de cartões perfurados e uma nova forma

foi necessária. E no meio da década de 50 os primeiros OCRs surgiram comercialmente.

5.1.1 Primeira geração

Os OCRs que surgiram entre 1960 e 1965 podem ser considerados a primeira geração de OCR. Eles basicamente restringiam os formatos das letras a serem reconhecidos. Os símbolos eram especialmente projetados a leitura pelas máquinas e não pareciam nada natural aos olhos do ser humano. Com o passar do tempo máquinas multifontes, que podiam ler até dez fontes, começaram a aparecer. O número de fontes era limitado pelo método de reconhecimento usado, o template matching, que compara com a imagem do caracter com uma biblioteca de protótipos.

5.1.2 Segunda geração

A segunda geração apareceu entre a metade da década de 60 e começo da de 70. Eles eram capazes de reconhecer caracteres impressos por máquinas e também escritos à mão. No entanto os caracteres reconhecidos quando escritos à mão era restrito a algumas letras e símbolos.

Nesse período foi feito um grande esforço para a padronização. Em 1996 o conjunto de caracteres utilizados para OCR foi definido: o OCR-A. Essa fonte era muito estilizado e projetado para facilitar o reconhecimento de caracteres, porém legível para o ser humano. Um padrão europeu foi desenvolvido, o OCR-B, que era uma fonte mais natural que o padrão americano. Foram feitas algumas tentativas para tentar unir esses dois padrões, mas ao invés disso surgiram máquinas capazes de reconhecer os dois padrões.

5.1.3 Terceira geração

O desafio enfrentado pela terceira geração dos OCRs foi documentos com pouca qualidade e grande quantidade e conjunto de caracteres escritos à mão. Baixo custo e alta performance eram objetivos importantes, que foi ajudado pelo grande avanço no hardware.

Mesmo que OCRs mais sofisticados tenham começado a surgir, o OCR mais simples ainda era muito utilizado. Na época os computadores pessoais e as impressoras a laser ainda não dominavam a área de produção de texto. A maioria dos textos eram produzidas por máquinas de escrever. A uniformidade nos espaçamentos e o pequeno número de fontes tornava o trabalho dos OCRs mais fácil. O texto poderia ser criado por digitadores e inserido no computador através do OCR para a edição final. Dessa forma, o editor de

texto, que era um recurso muito caro na época, poderia ser usado por várias pessoas e os custos poderiam ser cortados.

5.2 Métodos para reconhecimento de caracteres

O reconhecimento de caracteres é um subconjunto da área de reconhecimento de padrões. No entanto, foi devido ao incentivo do reconhecimento de caracteres que fez com que o reconhecimento de padrões e análise de imagens se tornassem o que são hoje.

O princípio do reconhecimento automático de padrões é primeiro ensinar a máquina que classes de padrões que podem aparecer e como eles são. No OCR, os padrões são letras, números e alguns símbolos especiais. O ensino é feito mostrando à maquina exemplos de caracteres de todas as diferentes classes. Baseando-se nesses exemplos, a máquina constrói um protótipo ou uma descrição de cada classe de caracteres.Durante o reconhecimento, os caracteres desconhecidos são comparados com as descrições obtidos anteriormente e tenta achar aquele mais próximo.

Em muitos OCRs o processo de treino ocorre antes de chegar ao usuário final, enquanto que em alguns a inclusão de novas classes de caracteres é possível.

5.2.1 Componentes de um OCR

Um OCR consiste em vários componentes. O primeiro passo é digitalizar o documento analógico usando um scanner óptico. Quando a região contendo texto é localizada, cada imagem do símbolo é extraído através de um processo de segmentação. A imagem do símbolo extraído pode ser pré-processado para a eliminação de ruídos, para facilitar a extração dos aspectos no próximo passo.

A identidade de cada símbolo é encontrada comparando os aspectos extraídos da imagem com descrições dos caracteres das diferentes classes, obtidas na fase de ensino.

5.2.2 Scanner Óptico

A imagem digital é obtida através de um scanner óptico que, de forma simplificada, consiste em um equipamento que converte as intensidades de luz em escalas de cinza. Como geralmente os documentos são impressos em papel branco com letras em preto, é muito comum os OCRs converterem a imagem

em escala de cinza em uma imagem binária, formada por preto e branco, aplicando uma função chamada threshold. Nesse processo é definido um valor de threshold e todos os níveis de cinza abaixo dele é transformado em branco e os acima dele são transformados em preto. Para documentos com variação do nível de cinza muito grande esse método pode ser insuficiente. Por exemplo, um documento onde parte dele está manchado com algo que torne o fundo um pouco mais cinza. Se escolhido o valor de threshold errado, toda a parte da mancha pode ser ignorado pelo OCR.

Portanto o ideal é utilizar um método de para threshold que é capaz de variar o valor de threshold de acordo com a região do documento.

5.2.3 Localização e segmentação

Segmentação é o processo que determina os elementos de uma imagem. É necessário localizar as regiões do documento onde o texto foi impresso e distinguir deles de figuras e gráficos.

A segmentação no texto é isolar os caracteres ou palavras. A maioria dos algoritmos de reconhecimento de caracteres isola o caracter individualmente. Os principais problemas da segmentação pode ser divido em quatro grupos:

- **Extração de caracteres grudados ou fragmentados -** Tal distorção pode resultar que dois ou mais caracteres sejam interpretados como um só ou que um caracter seja identificado como vários.
- **Distinguir ruído de texto -** Pontos e acentos podem ser confundidos como ruído e vice-versa.
- **Identificar uma imagem ou figuras geométricas como texto -** Um elemento que não é texto vai ser mandado para o reconhecimento.
- **Identificar um texto como imagem ou figura geométrica -** Nesse caso um texto vai ser ignorado e ele não vai ser mandado para o reconhecimento.

5.2.4 Pré-processamento

A imagem digital resultante do processo de escaneamento pode conter ruídos. Dependendo de vários fatores como a resolução do scanner e da aplicação do processo de threshold, os caracteres podem ficar borrados ou com buracos. Alguns desses defeitos podem prejudicar muito o reconhecimento dos caracteres, e eles podem ser eliminados no pré-processamento para suavizar os caracteres.

A suavização implica em preencher e afinar. Ao preencher, pequenos buracos são eliminados e ao afinar reduz a largura da linha.

Além da suavização o pré-processamento geralmente inclui normalização. Ela é aplicada para obter caracteres de tamanho, inclinação rotação uniforme.

5.2.5 Extração do aspecto

O objetivo da extração do aspecto é capturar as características essenciais de um símbolo. A técnica para a extração dessas características são divididas em três grupos, onde os aspectos são encontrados em:

- Na distribuição dos pontos
- Transformações e expansões de séries.
- Análise estrutural

5.2.6 Classificação

A classificação é o processo de identificar cada caracter e associá-lo à sua classe de caracter correto.

Alguns métodos para a classificação:

- Métodos de decisão teórica
  - Casamento O casamento cobre os grupos de técnicas que se baseiam na mensuração da similaridade, onde a distância entre o vetor descritivo, que descreve o caracter extraído, e a descrição de cada classe é calculada.
  - Classificador estatístico ótimo Na classificação estatística um método probabilístico para o reconhecimento é aplicado. A idéia é utilizar o esquema de classificação que é ótimo, no sentido que em média ele dá a menor probabilidade de erro na classificação.
  - Redes Neural O uso da rede neural para reconhecimento de caracteres e outros tipos de padrões voltou com força. Essa rede é formada por várias camadas de elementos interconectados. Um vetor descritivo entra na rede transforma essa entrada em uma função não linear. Durante o ensino do OCR cada elemento da rede é ajustada para que a saída desejada seja obtida.
- Métodos estruturais

– Mede a similaridade baseando-se em conceitos gramaticais. A idéia é que cada classe tenha sua própria gramática. Por exemplo, suponha que exista duas classes que podem ser geradas pela gramática G1 e G2, respectivamente. Dado um caractere desconhecido, dizemos que ele é mais similar à primeira classe se ele pode ser gerado por G1, mas não por G2.

5.2.7 Pós-processamento

Agrupamento

O resultado do reconhecimento é um conjunto de caracteres individuais. No entanto, gostaríamos de associar esses símbolos individuais com outros, para formar palavras e números. Esse processo é chamado de agrupamento. Ele se baseia na localização deles no documento, símbolos considerados suficientemente pertos são agrupados.

Detecção de erro e correção

Um sistema que faz o reconhecimento de um único caracter, desconsiderando todo resto não é o suficiente. Mesmo os melhores sistemas de reconhecimento não conseguem atingir 100% de acerto. Porém muitos desses erros podem ser detectados e/ou corrigidos.

Existem dois principais métodos, onde o primeiro analisa a possibilidade de seqüências de caracteres aparecerem. Isso pode ser feito usando regras que definem a sintaxe da palavra, falando por exemplo que depois de um ponto final, normalmente vem uma letra maiúscula. Para diferentes línguas, a probabilidade de dois caracteres aparecerem em sequência pode ser usada para a detecção de erro. Por exemplo, na língua inglesa a probabilidade de aparecer a letra "k" depois de um "h" em uma palavra é zero. Se uma combinação dessas for encontrada pode-se assumir que um erro foi encontrado.

Outro método é o uso de dicionários. Dado uma palavra, o sistema faz uma busca no dicionário. Se não encontrá-lo, assumimos que foi detectado um erro e ele pode ser corrigido mudando a palavra para outro mais próximo existente no dicionário. Ao encontrar uma palavra no dicionário não prova que ele não tenha erro. O erro pode ter transformado uma palavra em outra que existe no dicionário.

O problema desses métodos é que eles não são aplicáveis em todos os contextos. Por exemplo, podemos estar reconhecendo um texto em português, mas podemos misturar palavras de outras línguas, que não existem no dicionário. Outro caso é a criação de palavras em uma língua. Com o passar dos

tempos, novas palavras são inventadas e inseridas na língua, e tais palavras não estão no dicionário.

Capítulo 6

Atividades realizadas

A primeira parte do sistema de GED que estudei foi a de reconhecimento de caracteres. O sistema utiliza um componente proprietário que efetua o reconhecimento. Minha primeira tarefa foi estudá-lo e aprender como utilizá-lo. Tive que estudar um pouco mais a fundo como era feito o reconhecimento dos caracteres para poder entender quais eram os pontos fracos do reconhecimento, e assim tentar maximizar a taxa de acerto.

Meu segundo trabalho foi estudar a certificação digital. Tive que pesquisar e entender todo o processo e estrutura utilizada. Feito isso meu próximo passo foi implementar um software que assinava e verificava assinatura de documentos. Como se tratava de segurança dos dados, os cuidados foram redobrados.

Até esse momento eu estava trabalhando em módulos isolados do sistema, sem me preocupar com a parte web da aplicação. Portanto o meu próximo passo foi estudar como as aplicações web são e como o sistema estava implementado. Ele utiliza o framework Struts e portanto tive que estudá-lo a fundo.

Após isso, trabalhei com uma reestruturação feita no workflow no sistema.

Capítulo 7

Conclusão

Após esse estágio o resultado e o produto obtido foi um sistema GED com muito mais funcionalidade. Esse estágio foi muito bom pois possibilitou estudar várias áreas da computação, sem ficar muito preso a um tema.

Referências Bibliográficas

[1] Johnson, Ralph E. Components, Frameworks, Patterns

[2] Fayad, Mohamed Schmidt, Douglas C.Object-Oriented Application Frameworks

[3] Spielman, Sue The Struts Framework - Practical Guide for Java Programmers

[4] Struts. http://struts.apache.org, Novembro 2006.

[5] Designing Enterprise Applications with the J2EETM Platform, Second Edition.http://java.sun.com/blueprints/guidelines/ designing_enterprise_applications_2e/DEA2eTOC.html, Novembro 2006.

[6] Struts, an open-source MVC implementation.http://www-128.ibm.com/developerworks/ ibm/library/j-struts, Novembro 2006.

[7] Stamp, Mark. Information Security - Principles and Practice. John Willey Sons, 2005.

[8] ICP-Brasil. http://www.icpbrasil.gov.br, Novembro 2006.

[9] Instituto Nacional de tecnologia da informação. http://www.iti.gov.br/, Novembro 2006.

[10] Terada, Routo. Segurança de Dados - Criptografia em Redes de Computador. Editora Edgard Blücher LTDA, 2000.

[11] Gonzalez, Rafael C., Woods, Richard E. Digital Image Processing. Prentice Hall, 2002.

[12] Kharma, Nawwaf N., Ward, Rabad K. Character Recognition for Non Experts.

[13] Eikvil, Line. Optical Character Recognition.

Parte II

Parte Subjetiva

7.1 Desafios e frustações encontradas

Resolvi fazer o estágio pois no curso senti falta da parte prática. Queria aprender como as coisas eram fora do mundo acadêmico, e aprender coisas novas.

O grande desafio que tive foi talvez gerenciar o tempo dedicado ao estudo do assunto com o tempo dedicado à implementação. Como a utilização de componentes e frameworks é muito intensa, fica difícil saber até que ponto é possível se aprofundar no assunto sem prejudicar no cronograma existente.

Uma coisa que percebi no estágio foi que tem muita coisa errada no processo de desenvolvimento de software nas empresas. Isso foi uma frustração no sentido que eu esperava aprender essas coisas durante o estágio, mas o que percebi foi que eu deveria estudar e propor soluções. Ao mesmo tempo isso foi grande desafio.

7.2 Disciplinas relevantes para o trabalho

As disciplinas mais relevantes para o trabalho foram:

- MAC110 - Introdução à computação
- MAT111, MAT121 e MAT221 - Cálculo Diferencial e Integral I, II e IV.
- MAE121 e MAE212 - Introdução à probabilidade e estatística.
- MAT138 - Álgebra I para computação.
- MAC122 - Princípios de desenvolvimento de algoritmos.
- MAT139 - Álgebra linear para computação.
- MAC211 e MAC242 - Laboratório de programação I e II.
- MAC323 - Estrutura de dados.
- MAC239 - Métodos formais em programação.
- MAC316 - Conceitos fundamentais de linguagem de programação.
- MAC328 - Algoritmo em grafos.
- MAC338 - Análise de algoritmos.
- MAC426 - Sistemas de Banco de Dados.

- MAC332 - Engenharia de software.
- MAC422 - Sistemas Operacionais.
- MAC438 - Programação concorrente.
- MAC336 - Criptografia de segurança de dados.
- MAC413 - Tópicos de Programação Orientada a Objetos
- MAC417 - Visão e Processamento de Imagens
- MAC441 - Programação Orientada a Objetos

7.3 Interação com membros da equipe

A interação com os membros da equipe foi muito boa. Não tive dificuldades de apresentar os meus problemas e dúvidas e os meus mentores sempre tiveram disponibilidade para me atender.

7.4 Observações sobre aplicações dos conceitos na prática

Nas empresas muitas vezes não temos tempo de nos aprofundarmos muito nos estudos em um tema. Já existem muitos componentes prontos que são utilizados com muita freqüência. Muitas vezes, entendido o básico sobre o assunto devemos já partir para a implementação, pois o cronograma é apertado. As outras coisas necessárias são aprendidas durante o processo de desenvolvimento, e isso muitas vezes gera mais atraso.

7.5 Passos futuros

Pretendo agora estudar mais para melhorar o processo de desenvolvimento de software da empresa. O processo atual tem muitas falhas e é ineficiente. A solução não é tão simples e talvez eu necessite um pouco mais de experiência para que eu possa chegar a um ponto perto do ideal.

O meu plano é estar sempre estudando, não só a parte da computação mas também outras áreas de interesse como administração e outras.

Estou considerando em fazer um mestrado, mas não no ano de 2007. Quero explorar mais o mundo empresarial e aprender muito sobre isso. Além disso, tem muitas coisas ainda que quero colocar em prática.